Infiniviz: Taking Quake 3 Arena on a Large-Scale Display System to the Next Level

Rudolfs Bundulis, Guntis Arnicans
University of Latvia
Riga, Latvia
rudolfs.bundulis@lu.lv, guntis.arnicans@lu.lv

Abstract—**The authors of this paper have previously presented a large-scale display system called Infiniviz in other publications. Infiniviz attempts to improve network bandwidth consumption and computational performance compared to other existing large-scale display systems. Since the previous publications have been made in early development stages of Infiniviz, only the overview of the software architecture and details of hardware implementation have been presented so far. This paper contains a real-life test of Infiniviz running Quake 3 Arena at a resolution of 9600 x 5400 at 24fps. Also, in this paper, the authors have tried to match their results to what has been published by other researchers to provide the foundation for the claims of improvement.**

I. Introduction

Leveraging the display limits imposed by hardware or software limitations of a single computing system is one of the key factors driving the research in the area of large-scale display systems. The main focus in particular research can differ – some systems focus on ease of collaboration, others on maximizing the overall result regarding display resolution, fps, and physical size. However, all of the research mainly encounters similar problems.

Let us define a *large-scale display system* as a system with two distinct components - the content producer and the content visualizer.

The *content producer* can be anything from a single piece of software or hardware (most commonly a standard PC system) to an interconnected network of such or more complex entities. A good example of the latter case would be a scenario where a group of people wants to collaborate by showing the content of their PCs on a virtualized large-scale display.

The *content visualizer* is a system capable of accepting some kind of logical or rasterized representation of the content and displaying it in high resolution (higher than what can be achieved with a conventional GPU).

With such definition, we can also look at a standard PC system as a large-scale display system, where the content producer is the operating system and the content visualizer is the GPU. In such case the system is usually limited by the maximum performance of the GPU, the number of outputs on the GPU and their resolutions.

Historically the visualization capabilities have been the ones imposing the limits. Initially, this issue has been tackled by creating more complex visualization components to be able to handle the produced content. This indirectly imposes a need to create logic to expose this complex display system to the producer in a standardized and convenient way.

The second chapter of this paper covers existing research in the large-scale display area. The authors of this paper chose the most cited and referenced publications, thus not all existing research is mentioned. The chapter briefly summarizes how each of the researchers has tried to solve the complexity issues and overcome the performance limitations.

In the third chapter, the authors present their own large-scale display system called *Infinviz*. The chapter contains a brief description of the architecture and two hardware setups that Infiniviz has been run on so far.

The fourth and fifth chapters contain performance measurements of Invinifiz running Quake 3 Arena at a resolution of 9600 x 5400 and analysis of the obtained results.

The sixth chapter contains conclusions regarding the obtained results and other published research.

II. Large Scale Display Systems

To understand the key challenges present in the area of large-scale display systems we should first look at the existing research.

Deep View [1] was an attempt to overcome the maximum display resolution of 1600 x 1200 pixels at that time (2002) by creating a cluster of rendering nodes connected to a tiled display driven by IBM's Scalable Graphics Engine (SGE). SGE was a custom hardware system capable of accepting data over gigabit ethernet connections and rendering that data to a digital signal on DVI ports. To achieve the desired transparency to the content producer, the authors used *Chromium* [2], an OpenGL implementation that is exposed as a standard OpenGL driver to the content producer but internally divides the workload on to a clustered set of rendering nodes to achieve greater performance. The drawback was that this system could only visualize data coming from compatible content producers.

SAGE [3] allowed any software to stream pixel data to a virtualized large-scale display by using a custom library named SAIL (Fig. 1). It can be argued whether such approach where each possible software that wants to use SAGE needs to integrate with SAIL is better or worse than allowing existing OpenGL applications to seamlessly work with a large-scale display system like in Deep View. Also, the DXT pixel compression used in SAGE has a fixed ratio of 6:1, which is smaller than the compression ratio of popular image

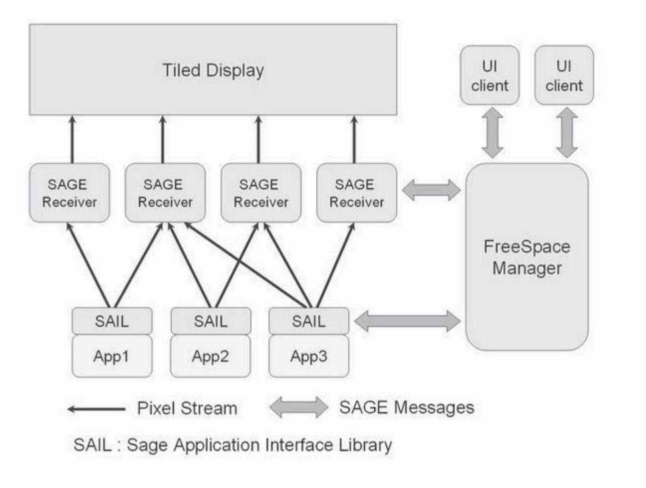

Fig. 1 SAGE architecture

compression codecs like JPEG. But compared to JPEG, DXT requires less computational power. Thus it is hard to state that the choice of DXT was good or bad since one can always argue whether the computational power required for the compression is more or less important than the required network bandwidth. Even with these points open for debate, SAGE is a very important step in the area, since it was the first large-scale display system with a major focus on collaboration. Since the architecture natively allows multiple pixel sources to be plotted on a virtualized display simultaneously, it creates a natural environment for collaboration. This inspired and drove a significant amount of further collaboration focused research.

The authors of *CGLX* [4] argued that both SAGE and Chromium are not scalable enough because SAGE relies on a high bandwidth network to transmit the pixel data and Chromium due to the ineffective workload distribution due to the sort-first approach and the logical representation of data in the OpenGL command stream. The sort-first approach distributes the rendering workload to rendering nodes responsible for the tile that will display the content. However, there are many cases where dynamic changes in content can be handled more efficiently by splitting the rendering of the content for a single tile to multiple parallel rendering nodes and then reassembling it together. The paper shows significant performance improvement over Chromium. Nevertheless, CGLX requires applications to integrate with their client APIs, even though as stated by the authors adapting existing OpenGL applications is easy.

Equalizer [5] was an attempt similar to CGLX. The authors also pointed out that the rendering workload distribution and data representation in Chromium might not be optimal. The provided results were again better than Chromium in the given test cases.

The authors *DisplayCluster* [6] tried to prove that the approach used by SAGE is somewhat better since it creates a natural windowing and collaboration environment – Chromium, CGLX, and Equalizer are not limited to mostly focus on a single content producer. Meanwhile SAGE allowed presenting content from multiple sources in parallel on a single homogeneous surface. DisplayCluster tried to solve the

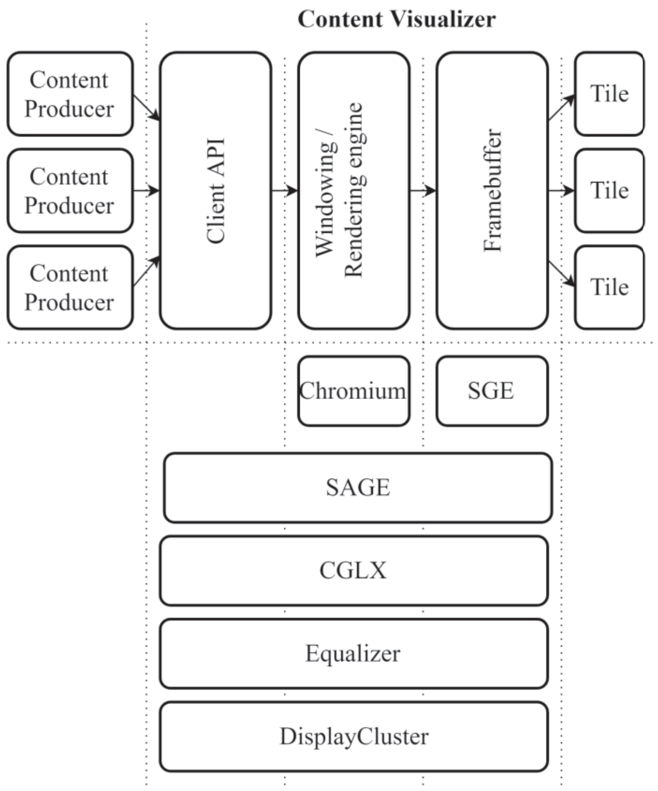

Fig. 2 Categorization of existing large-scale display systems

bandwidth requirements of SAGE by utilizing a SIMD accelerated JPEG compression.

The large-scale display systems mentioned above can be categorized as seen in Fig. 2. Generally, they all either provide their own or integrate with existing 2D / 3D drawing/rendering APIs and provide rendering and/or windowing engine.

The main reason for mentioning this existing research in the large-scale display system area in the context of this paper is to point out the complexity of the content visualizer component. In the case of Deep View, this was a network of rendering nodes connected to the IBM's SGE. In the case of SAGE, this was as a system consisting of SAGE receiver nodes, a free space manager and the SAIL driven client applications. CGLX, Chromium, and Equalizer utilized a rendering cluster that required efficient workload distribution algorithms and optimized the logical representation of the content. DisplayCluster provided its own windowing engine.

Moreover, the main need for such complexity was to overcome the limitations of the GPUs inside the content producers.

Currently, this balance has shifted - the GPUs are becoming more and more powerful. In the same time, hardware evolution and virtualization allow escaping the thought paradigm where a display system is limited to a few GPUs. New technologies allow moving the GPUs to the content producer component leaving the content visualizer with the sole task to efficiently present the content regarding FPS and bandwidth required for the content.

This trend can be seen in several recently developed large-scale display systems. The *Reality Deck* [7] is a 1.5 gigapixel display surface consisting of 18 hardware nodes each with 4 GPUs, most of which have 24 displays connected. Even though Reality Deck used Equalizer and other visualization frameworks for sending the content over to the display nodes from the head node it is still important that the system had no complex, nonstandard hardware components.

The work done by Kim et al. [8] addresses the issue of efficiently managing multiple GPUs in a large scale display system. The main focus of the work is to address the issue of parallel rendering operations decreasing the performance of the GPU. Such research further enables the use of standard PC systems with multiple GPUs to drive large-scale displays. If for example there was a system with a GPU that could drive an unlimited amount of displays to create a large homogeneous display surface it could suffer from multiple client applications drawing their separate portions with very short intervals. The issue arises from the traditional implementation of window managers in operating systems, where only a single GPU is used to render content. The authors address this issue by creating a custom client-side driver which abstracts the underlying GPUs. This client-side driver together with a server-side driver and a window compositor allow splitting the content produced by the clients efficiently to the GPUs responsible for displaying the particular area.

From the quick overview given previously, we can outline several key challenges that have driven the research in the area of large-scale display systems throughout the time.

Limitations imposed by the connection media. As stated by the IBM research report [9] one of the reasons for the development of IBMs SGE was a limitation on DVI cable length. Thus, raw pixel data transmission over Ethernet was deemed to be more effective. Further on SAGE tried to optimize the needed ethernet bandwidth by using a smarter way of representing the pixels. An alternate solution to reduce the ethernet bandwidth was to transmit a logical representation of the content – e.g., OpenGL texture data and rendering calls as done by Chromium. The authors of the Equalizer found the approach of Chromium to be suboptimal and made their own custom representation of the content data. Citing their paper: "*The problem comes in when the OpenGL stream is large in size, due to not only containing OpenGL calls but also the rendered data such as geometry and image data. Only if the geometry and textures are mostly static and can be kept in GPU memory on the graphics card, no significant bottleneck can be expected as then the OpenGL stream is composed of a relatively small number of rendering instructions. However, as it is typical in real-world visualization applications, display and object settings are interactively manipulated, data and parameters may change dynamically, and large data sets do not fit statically in GPU memory but are often dynamically loaded from out- of-core and/or multiresolution data structures. This can lead to frequent updates not only of commands and parameters which have to be distributed but also of the rendered data itself (geometry and texture), thus causing the OpenGL stream to expand dramatically. Furthermore, this stream of function calls and data must be packaged and broad- cast in real-time over the network to multiple nodes for each rendered frame. This makes CPU performance and network bandwidth a more likely limiting factor. While preserving a minimally invasive API, the novel proposed system is better aimed at scalability as the actual data access is decentralized in the distributed rendering clients.*"

Seamless integration. Large-scale display systems which are not able to display content from software running on standard operating systems would require lesser their adoption. Exposing them via standardized APIs like OpenGL instead of custom libraries enables transparency from the point of view of the content provider. Even though as mentioned before there are solutions that act as an OpenGL driver most large-scale display systems still require software integration of their custom client libraries.

Efficient use of hardware. Initially to overcome the performance bottlenecks of the GPUs researchers tried to use clustered rendering systems, as shown by the Deep View and the Reality Deck. One can argue, that such systems may suffer from underutilization of individual nodes. In contrast, research to allow efficient use of multiple GPUs inside a single hardware system has been done. As a result, software solutions that allow abstracting them and efficiently distributing the tasks among them have been developed. Such solutions enable easily turning existing computer systems into large-scale display systems.

As noted earlier, limitations imposed by GPUs have been part of the factors driving the research in the area of large-scale display systems. However, in parallel to the scientific research, GPU vendors have also addressed the same issue. For example, the fact that traditionally GPUs have a fixed number of physical display outputs and there is a limited amount of GPUs that can be fitted into a single motherboard, the total number of outputs for a given PC is limited. Since in most cases the number of physical display outputs usually increases together with the overall power and price of the GPU, in a scenario where a large display surface with a high resolution is needed for non-expensive rendering operations, a lot of GPU resources may end up unutilized. Thus, GPU vendors have expanded their products with different virtualization technologies, for example – NVIDIA vGPU. NVIDIA vGPU is mostly targeted to provide multiple virtual GPUs within a single physical GPU to allow efficiently utilizing a single GPU for multiple workstations in the form of virtual machines. Still, NVIDIA vGPU can also be used to create a single vGPU with a large number of virtual display outputs without actually having a single physical display output, thus removing the hardware limitations mentioned above.

III. Infiniviz

A. Introduction to Infiniviz

The authors of this paper have previously presented a large-scale display system named Infiniviz [10][11] in their previous publications. It differs from other systems described in the introduction by achieving a fully seamless integration with existing software.

Infiniviz is built upon virtualization. The architecture of Infiniviz is given in Fig. 3. A host Windows/Linux or MacOS operating system can run the Infiniviz software stack which internally uses VirtualBox to run the any required operating system that acts as the content producer. VirtualBox exposes a

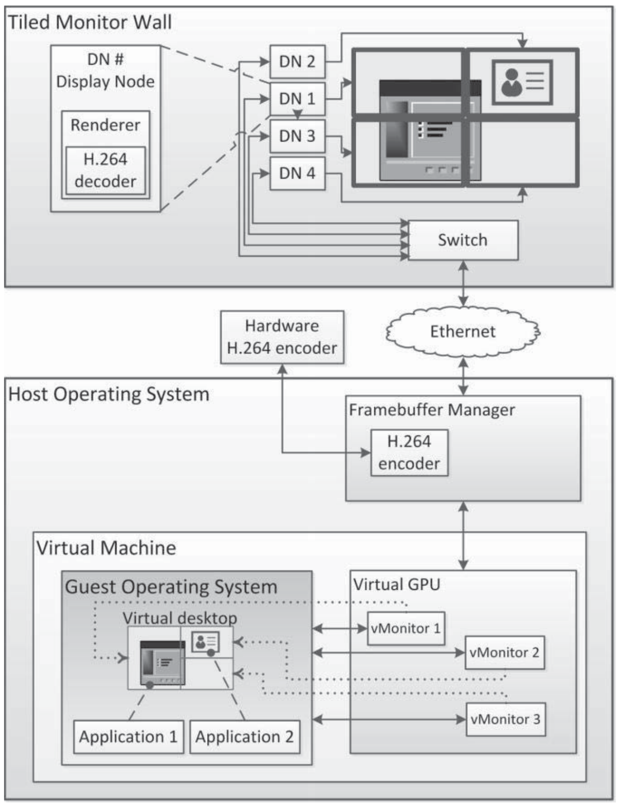

Fig. 3 Infiniviz architecture [11]

virtualized GPU to the guest operating system which matches the resolution and size of the physical tiled display surface. The authors initially compared VirtualBox with virtualization solutions available from the GPU vendors [12], but chose VirtualBox due to the fact that it was open-source and provided the largest overall resolution for the virtualized display surface. The Infiniviz software stack splits the content rendered in the framebuffer of the virtualized GPU into tiles corresponding to the physical tiles of the display surface. Each of these parts is then encoded into a video stream with the H.264 video codec and sent over to a Raspberry Pi unit, attached to the physical display. The Raspberry Pi unit decodes the stream and renders it onto the physical display.

The existing publications in the area contain debates on whether a sort-first or sort-last approach is better, and for what use cases. *Sort-first* means that the image is first split to individual workloads (usually corresponding to the tiles) and then rendered and/or encoded. *Sort-last* means that the partition of the work in workloads is based on contents (e.g, rendering an equal amount of primitives and then combining the final image for each tile from possibly multiple sources). Infiniviz implements a sort-first approach, which is most suited for the scenario where an already compressed content is sent to the rendering nodes.

To reduce the resource consumption of the video encoding Infiniviz utilizes hardware-accelerated video compression. Currently, NVIDIA NVENC and Intel Quick Sync are supported. The authors of this paper previously already published research comparing hardware accelerated H.264 encoding to CPU based H.264 encoding with FFmpeg and the algorithms used in other large-scale display – DXT (SAGE) and JPEG (DisplayCluster) [13].

Such approach has several benefits. Due to virtualization, Infiniviz requires no software integration. Any existing software can be run inside the guest operating system using the native drawing APIs. There are limitations on Direct3D and OpenGL version support, but that is something that can be added to VirtualBox to achieve full compatibility with any software. Next, by using hardware accelerated H.264 encoding the conversion of pixel data to a video stream with much lower bandwidth does not require much CPU resources thus reducing the computational overhead of the system and allowing the guest operating system to claim more resources. By using RTP protocol for the transmission, the Raspberry Pi display nodes can be replaced with other either embedded systems or RTP display capable TV sets. Thus, such design enables more flexibility in deploying Infiniviz in existing environments.

B. 3D acceleration in Infiniviz

A significant amount of content that requires the use of large-scale display systems is 3D based - various kinds of real-time 3D simulations, CAD tools, etc. Thus, the interoperability of the large-scale display system with industry standard 2D/3D APIs is an important factor in its design. As already stated in the paper, several other large-scale display systems have been implemented as OpenGL middleware by exposing themselves to the content provider software as a standard OpenGL driver or advertise easy transformation of existing OpenGL based software to their client APIs. OpenGL is popular due to its portability across operating systems.

Nevertheless, Direct3D is also a major API which has not been targeted by large-scale display systems so far. Since Infiniviz uses VirtualBox for virtualization of the guest operating system the content provider software, it can use the 3D API virtualization capabilities built in VirtualBox. Currently, VirtualBox supports Direct3D 8/9 and OpenGL 2.1 (given the underlying hardware supports at least OpenGL 2.1). Both APIs are forwarded to the underlying hardware – Direct3D by being translated to OpenGL and OpenGL - in a direct manner. An interesting fact is that VirtualBox uses Chromium internally to dispatch the OpenGL calls from the virtualized GPU to the host GPU. However, again, this is simply the current state of implementation in VirtualBox and can be subject to change when/if better alternatives are available.

For example, this means that Infiniviz can run Quake 3 Arena, which was also one of the features proudly presented by the authors of the Deep View. At the time Deep View was developed, the standard display resolution was 1600 x 1200 (1.92 Megapixels), and the Deep View allowed running Quake 3 Arena at a resolution of 3840 x 2200 (8.45 Megapixels). Infiniviz can run Quake 3 Arena at a resolution of 9600 x 5400 (51.8 Megapixels) with an average of 24 FPS.

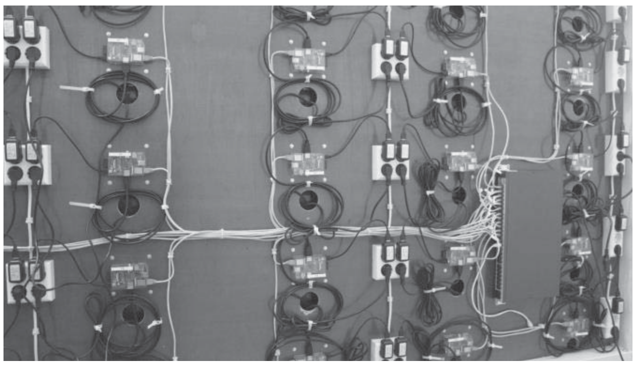
Fig. 4 LAN inter-connected Raspberry Pi devices at the back of the display wall

Fig. 5 Display wall server - Gigabyte Brix Pro mini PC

B. Infiniviz prototypes

After defining the architecture the authors proceeded to build an actual working system, that would be able to run Infiniviz. A tiled display wall consisting of 25 22” DELL LCDs arranged in a 5x5 matrix was constructed. Each of the displays was driven by a Raspberry Pi model B. All the Raspberry Pi devices are connected with a server that runs the Infiniviz software stack via an HP Gigabit Ethernet switch (Fig. 4).

Initially, the authors assembled a server that was meant for displaying mostly static content and did not require significant investment in hardware [10]. The server running the Infiniviz software stack was an Intel Core i7 4770R CPU (4 physical cores or 8 virtual cores at 3.2 GHz) with Intel Iris 5200 Pro GPU, 12 GB of RAM and Windows 8.1 (Fig. 5). The authors used Intel Quick Sync as the hardware accelerated video encoder.

Authors successfully used common (e.g., Google Chrome, PDF viewer) and domain-specific (e.g., video surveillance) desktop software applications thus demonstrating that this approach does not force software developers to write display wall aware software – everything that works on a desktop PC works the same way in the virtualized display wall. For example, the authors used Chrome web browser to visualize large graphs that would not fit in a standard 1920x1080 display (Fig. 6).

However, due to the limitations of Intel Quick Sync and the fact that it shares resources with the CPU, the frame rate was low in scenarios where the content was redrawn very often or the software producing the content required significant CPU resources.

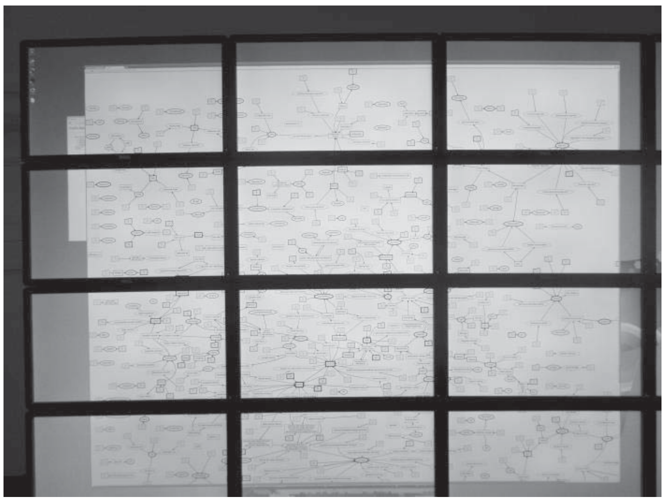
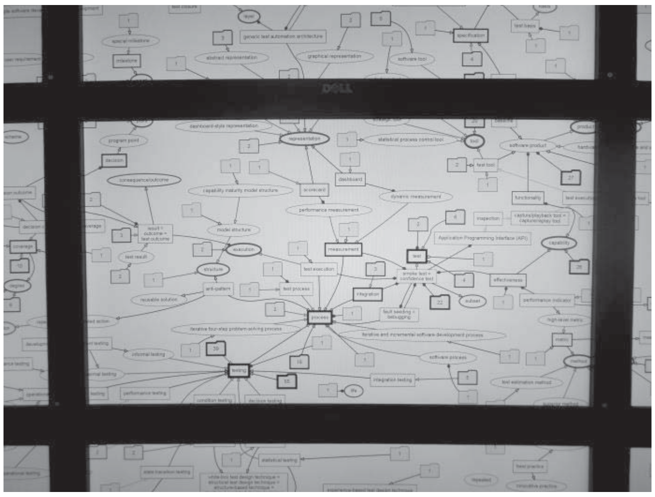
Fig. 6 Browser based visualization of large graphs inside a VM running on Infiniviz

Thus the authors moved on to evaluate NVIDIA NVENC, which is an H.264 encoder running solely in the GPU. This would prevent the video encoding from taking away CPU resources from the virtualized OS. Authors also decided to move away from Windows to Linux as the host platform because of integration problems with VirtualBox found on Windows.

Due to the massive parallelism in the Infiniviz software, the CPUs were upgraded to support more parallel encoding threads. The second Infiniviz server has 2 Xeon e5-2630 2.60 GHz CPUs (12 physical cores or 24 virtual cores) and two NVIDIA Quadro P4000 GPUs used for the video encoding. Only one of the GPUs was being used for the actual OpenGL rendering workload. This system is used for all current research on Infiniviz and the measurements presented further on in this paper.

IV. INFINIVIZ PERFORMANCE MEASUREMENTS

Given the major dominance of 3D based content in the publications presenting other large-scale display systems, the authors of this paper chose to use Quake 3 Arena as test benchmark. Quake 3 Arena was also chosen to support the claim of seamless software integration – this demonstrates that Infiniviz can run an existing desktop software without modifications. Also, this choice should enable existing and future research to produce comparable measurements. Since Quake 3 Arena is OpenGL based it should be out-of-the-box compatible with the systems exposing themselves as OpenGL drivers. The source code of Quake 3 Arena has been released under the GPL license by Id Software thus it can also be modified to support the systems with custom client APIs. Moreover, a sentiment for Deep View also played some role.

The test run was conducted by starting a Windows 7 guest virtual machine with a virtualized GPU at a resolution of 9600 x 5400. This resolution matches the physical tiled display wall available at the premises. The tiled wall consists of 25 22" DELL LCDs arranged in a 5x5 matrix (Fig. 7).

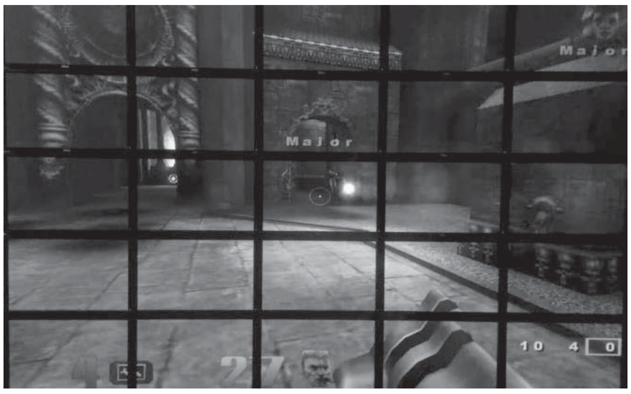

Fig. 7 Quake 3 Arena inside a VM running on Infiniviz

The authors added profiling functionality on both the server and the render nodes which collects data samples of CPU usage and network bandwidth with an interval of one second.

The actual test run, results of which are presented in the next chapter, consisted of 1) launching the Windows 7 virtual machine, interacting with mouse to launch Quake 3 Arena (Fig. 8, time segment A), 2) selecting a map (Fig. 8, time segment B), 3) playing for a short period of time (Fig. 8, time segment C), and 3) then exiting and shutting down the machine (Fig. 8, time segment D). A total of 170 data samples covering around two and a half minutes were collected.

V. RESULTS

Figure 8 presents the performance measurements gathered during the test run described in the previous chapter. Both frame rate and resource consumption metrics are available.

The upper chart shows frame rates during the test run. Infiniviz was configured to run at a fixed frame rate of 25 frames per second per tile. This means that if VirtualBox did update a region corresponding to a single tile more frequently than in 40ms, only the last changes would be encoded. This configuration explains the difference between both of the lines – in some regions of the graph the VirtualBox frame rate is higher because the updates were more frequent than 25 frames per second.

On the other hand, if no changes were made by VirtualBox, no new frames were encoded to avoid wasting bandwidth or computational resources. This explains the low frame rates on the left side of the chart since while Windows was loading and only mouse and keyboard interaction was used to locate Quake 3 Arena and launch it, updates by VirtualBox were infrequent.

Overall the fact that the lines are quite close shows that the Infiniviz software was performing quite well in the given test. Authors also used the built-in functionality of Quake 3 Arena to display the frame rate from inside the game. Quake 3 Arena reported an average of 24 frames per second. With 25 tiles to encode the video for this would give an average of 600 frames.

The upper chart shows that the actual frame encoding rate was quite close to this number, even though the drops and peaks show that further stability tuning is required. In an ideal case the line showing VirtualBox frame rate would have peaks and drops with an average of 600 frames (since the Infiniviz environment cannot control the software inside the virtual machine and how often it draws), but the line showing Infiniviz encoding rate would stay steady on 600 frames.

The lower chart shows network bandwidth (the bandwidth required for all 25 video streams) and CPU consumption during the test. The network bandwidth chart shows that with static content (Windows 7 desktop environment as well as the menu of Quake 3 Arena) the required bandwidth is low and would fit in even a 100 Mbit ethernet network. Thus, Infiniviz could serve as a remote collaboration environment where the whole screen is not updated frequently. When the game starts the rendering loop, the bandwidth raises to an average around 300 Mbps (12 Mbps per tile) with a peak at 325 Mbps. The CPU measurements show that even more CPU consuming software could be run inside the virtual machine since the CPU is not fully utilized.

One of the biggest issues that prevent the authors of this paper from a thorough comparison of the Infiniviz large-scale display system with other published research is the lack of diversified measurements in other published research. Even though, as stated in the introduction, there are at least two key factors – performance overhead and required transmission medium bandwidth – that the other large-scale display systems try to solve, only a few metrics are used to describe the achieved results. For an objective comparison among the large-scale display systems, measurements covering at least these two factors are required. Technical specifications of these systems should also be given precisely enough to allow scaling and comparing the performance results.

Another issue is the lack of a portable test environment. As mentioned in the introduction, while some of the large-scale systems act as drivers for the popular OpenGL API others require client software to integrate with their API libraries to

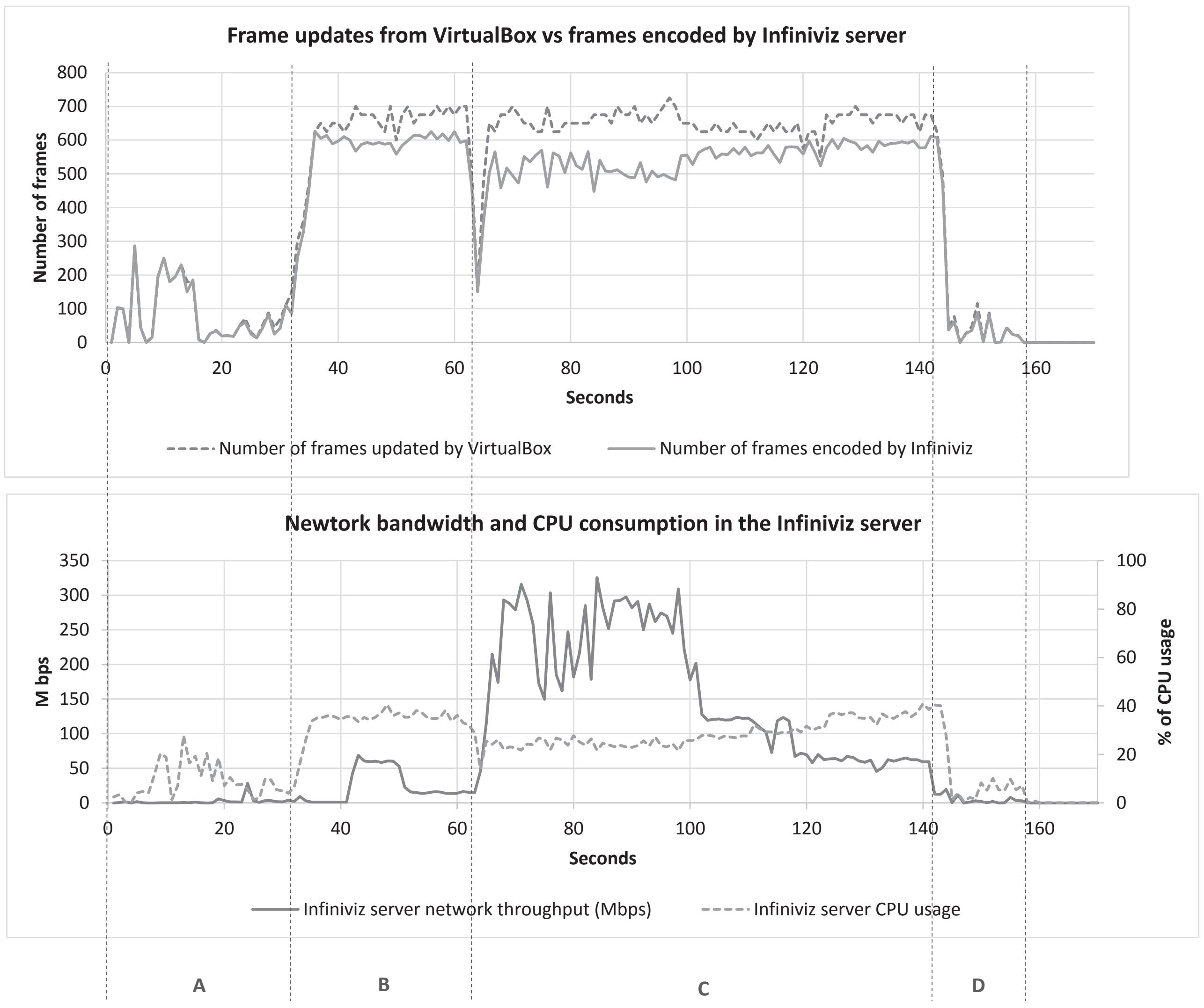

Fig. 8 Performance measurements

be able to present the content. Since OpenGL is a cross-platform API, a fixed test scene/software can be used to compare OpenGL compatible solutions on different platforms. Still, some authors prefer using browser-based WebGL tests, which make the comparison harder. Even though WebGL is a web browser-based abstraction over OpenGL, web browsers have implementation differences among them and even among several versions of the same browser. Thus, a very precise description of the test setup must be given for others to be able to produce comparable measurements.

Even if these constraints are met, it is not possible to compare such results with large-scale display systems with custom client API libraries. Thus, a mixed test environment that would be able to produce constant test content for OpenGL and the custom API libraries should be available. The authors of this paper will put further effort into developing a test environment able to cover as many of the published large-scale display systems as possible. Because currently, the lack of published performance results, is one of the main results to make strong arguments in favor of the Infiniviz architecture and its benefits in comparison with other large-scale display systems.

VI. Conclusion

In this paper the authors have presented through measurements of Infiniviz by running Quake 3 Arena demo with an averaged 24 FPS at a resolution of 9600 x 5400 inside a Windows 7 virtual machine in VirtualBox. Both CPU usage and network bandwidth usage during the test run have been provided. During the test run the authors sampled the CPU and network bandwidth usage data once per second on both the server running VirtualBox and encoding the video streams and on the Raspberry Pi systems rendering the streams once per second. The test run includes a full lifecycle of the virtualized operating system - startup, running Quake 3 and shutdown. This data should enable other researchers to perform further comparisons of their systems to Infiniviz.

One of the few systems that have enough published research data for the authors of Infiniviz to make the comparison is DisplayCluster. The system was built in 2008 and research was published in 2012. The authors of the publication stressed the vast improvement over SAGE regarding the required network bandwidth. This was mainly due to using lossy JPEG compression (DisplayCluster used SIMD accelerated libjpeg-turbo). In comparison, SAGE uses a lossy pixel based DXT compression algorithm with a fixed compression ratio 6:1 that was originally developed by S3 Graphics. Even though the authors of DisplayCluster have not published the test scenes or data used, they have provided network bandwidth requirements for given frame rates. The published results show that DisplayCluster was able to stream 12.5 FPS at a resolution of 48 megapixels with a network bandwidth of around 20 MB/s and 10.4 FPS at a resolution of 64 megapixels with a network bandwidth of around 26 MB/s.

The results presented in this paper cannot be directly compared to DisplayCluster since the resolutions do not match directly. However, if an approximation is used, Infiniviz shows better performance, since Infiniviz was able to display 52 megapixels with an average of 24 FPS under a 3D rendering load from Quake 3 Arena. Taking the 12.5 and 20 FPS values for 48 and 64 megapixels from DisplayCluster and approximating the value for 52 megapixels at 14.3 FPS yields better performance for Infiniviz. To perform correct bandwidth comparisons, a fixed test environment would be needed, since the size of the compressed video stream varies depending on the content. As can be seen from the measurement graphs (Fig. 8), the bandwidth required for all 25 compressed video streams from the Infiniviz server varies a lot depending on the content. The total consumed bandwidth ranges from around 50 Mbps (6.25 MB/s) when only Windows desktop and static windowed content is present to 325 Mbps (~40 MB/s) under 3D content from Quake 3 Arena. On the other hand, the results from DisplayCluster include only fixed bandwidth measurements per each resolution, which seems strange, since DisplayCluster is using lossy JPEG compression, thus intuitively the compressed stream size should also vary, similar to Infiniviz. Also, PSNR value measurements would be needed to evaluate two systems that employ lossy video compression like Infiniviz and DisplayCluster to perform the comparison at an equal picture quality level. Otherwise, pure bandwidth numbers are incomparable, since they could differ only due to changes in the picture quality.

ACKNOWLEDGMENT

The research was developed under the University of Latvia contract no. AAP2016/B032 “Innovative information technologies”.

REFERENCES

[1] J. T. Klosowski, T. James, J. Valuyeva, and G. Abram, “Deep view: high-resolution reality,” *IEEE Comput. Graph. Appl.*, vol. 22, no. 3, pp. 12–15, 2002.

[2] G. Humphreys, M. Houston, R. Frank, J. T. Klosowski, S. Ahern, and P. D. Kirchner, “Chromium : A Stream-Processing Framework for Interactive Rendering on Clusters,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 693–702, 2002.

[3] R. Luc *et al.*, “SAGE : the Scalable Adaptive Graphics Environment,” in *Proceedings of WACE 2004*, 2004.

[4] K. U. Doerr and F. Kuester, “CGLX: A scalable, high-performance visualization framework for networked display environments,” *IEEE Trans. Vis. Comput. Graph.*, vol. 17, no. 3, pp. 320–332, 2011.

[5] S. Eilemann, R. Pajarola, and M. Makhinya, “The Equalizer Parallel Rendering Framework,” *IEEE Trans. Vis. Comput. Graph.*, vol. 15, no. 3, 2009.

[6] G. P. Johnson, G. D. Abram, B. Westing, P. Navrátil, and K. Gaither, “DisplayCluster: An interactive visualization environment for tiled displays,” *Proc. - 2012 IEEE Int. Conf. Clust. Comput. Clust. 2012*, no. Figure 1, pp. 239–247, 2012.

[7] C. Papadopoulos, S. Member, K. Petkov, A. E. Kaufman, A. E. Kaufman, and K. Mueller, “The Reality Deck - Immersive Gigapixel Display,” *IEEE Comput. Graph. Appl.*, vol. 35, no. 1, pp. 33–45, 2015.

[8] I. Kim, J. Kim, J. Park, and Y. I. Eom, “Software-based Single-node Multi-GPU Systems for Interactive Display Wall,” *IEEE Trans. Consum. Electron.*, vol. 63, no. 2, pp. 101–108, 2017.

[9] I. B. M. F. D. Technology, S. L. Wright, and Y. Heights, “IBM 9.2-Megapixel Flat-panel Display: Technology and Infrastructure,” 2002.

[10] R. Bundulis and G. Arnicans, “Concept of virtual machine based high resolution display wall,” in *2014 IEEE 2nd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, 2014.

[11] R. Bundulis and G. Arnicans, “Infiniviz — Virtual Machine Based High-Resolution Display Wall System,” in *Databases and Information Systems IX: Selected Papers from the Twelfth International Baltic Conference, DB&IS 2016*, 2016, pp. 225–238.

[12] R. Bundulis and G. Arnicans, “Virtual Machine Based High-Resolution Display Wall : Experiments on Proof of Concept,” *Balt. J. Mod. Comput.*, vol. 5, no. 4, pp. 379–390, 2017.

[13] R. Bundulis and G. Arnicans, “Use of H.264 real-Time video encoding to reduce display wall system bandwidth consumption,” in *Advances in Information, Electronic and Electrical Engineering, AIEEE 2015 - Proceedings of the 2015 IEEE 3rd Workshop*, 2015.