OpenVideoWalls: an Open-Source System for Building Video Walls with Recycling Heterogeneous Displays

Zichen Zhu*
Rutgers University
Piscataway, New Jersey, USA
zichen.zhu@rutgers.edu

Zhongze Tang*
Rutgers University
Piscataway, New Jersey, USA
zhongze.tang@rutgers.edu

Amir Nassereldine
University at Buffalo
Buffalo, New York, USA
amirnass@buffalo.edu

Jinjun Xiong†
University at Buffalo
Buffalo, New York, USA
jinjun@buffalo.edu

Sheng Wei†
Rutgers University
Piscataway, New Jersey, USA
sheng.wei@rutgers.edu

Abstract

With the rapid development of the consumer electronics industry, how to efficiently recycle electronic waste such as display screens has become an important issue. Traditional display screen recycling methods are time-consuming and labor-intensive with a low recycling rate. Therefore, we propose an open-source and low-cost display screen recycling method, which enables everyone to assemble old displays into a larger display screen (i.e., video walls) and ensure premium quality even with distinct specifications among the multiple displays. In this paper, we focus on addressing the unique challenge of video walls that the video may be out of sync among multiple displays. Our proposed algorithm attaches a carefully designed timestamp to each frame on the server side and manages the frames based on the timestamps on the client side to achieve video synchronization. We deploy the synchronization algorithm in an end-to-end video wall system including a streaming server and a client (i.e., a video wall) consisting of multiple displays and their respective controlling Raspberry Pis. The evaluation results show that our algorithm effectively ensures video synchronization among different displays and can immediately resume synchronization after being affected by network fluctuations. Meanwhile, our algorithm does not introduce additional latency to the system or reduce video quality.

CCS Concepts

• **Information systems** → **Multimedia streaming**.

Keywords

video wall, e-waste recycling, display synchronization, video streaming

*Both authors contributed equally to this research.
†Corresponding Authors.

MMASIA '24, December 03–06, 2024, Auckland, New Zealand

ACM ISBN 979-8-4007-1273-9/24/12
https://doi.org/10.1145/3696409.3700251

ACM Reference Format:
Zichen Zhu, Zhongze Tang, Amir Nassereldine, Jinjun Xiong, and Sheng Wei. 2024. OpenVideoWalls: an Open-Source System for Building Video Walls with Recycling Heterogeneous Displays. In *ACM Multimedia Asia (MMASIA '24), December 03–06, 2024, Auckland, New Zealand*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3696409.3700251

1 Introduction

Electronic devices are increasingly short-lived with millions of screen displays dumped into the wasteland, causing significant harm to the environment. According to the Global E-waste Monitor report by the United Nations [8], approximately 5.9 billion kilograms of discarded screens and monitors from televisions, computers, and mobile devices were produced by the world in 2022. While only 22.3% of e-waste was reported as recycled, the recycling process itself presents significant challenges [25]. For instance, the recycling procedure of the displays is often resource-intensive, with the requirement of massive amounts of labor and energy. Moreover, the displays often contain various hazardous chemicals like arsenic, mercury, and cadmium [22], which are difficult to neutralize. These toxic matters pose significant risks to the environment and human beings if not handled properly. Therefore, we believe a better sustainable strategy is to find a novel solution to reuse those eliminated yet functional displays in a meaningful way.

One widely used application of displays is video walls [21], which can be deployed for retail [19], museums [15], entertainment [13], and public spaces [7]. For example, Times Square in New York City [7] is famous for its large video walls displaying advertisements and creative videos. Video walls provide a better user experience by combining multiple screens to construct a single large display. It would be great if we can build video walls using recycled and heterogeneous displays because it helps reduce e-waste in a less labor- and energy-intensive way.

Figure 1 shows the architecture of a representative existing video wall system with several components involved. The Video Source can be single or multiple image files, video files or live cameras connecting to the Video Wall Controller and providing the content to be displayed on the video wall. The Remote can change the layouts or switch video sources by sending infrared signals to the Video Wall Controller. The Video Wall Controller processes the video signals as desired and sends them to the corresponding displays on the Video Wall. A major issue with existing video walls

is that they are typically built with homogeneous displays with proprietary software systems, which are often costly and constrain customers in the choice of displays. The existing systems suffer from a lack of technical documentation and clearly defined universal metrics, making comparison and evaluation challenging for research and development. Additionally, the high cost of these systems further impedes their accessibility and development. Thus, we propose OpenVideoWalls, an open-source system with a set of system-independent metrics to help anyone to build their own video walls by reusing any used displays they may have, including different sizes, resolutions, and other parameters (e.g., supported refresh rates), and compare such similar systems under the same standard.

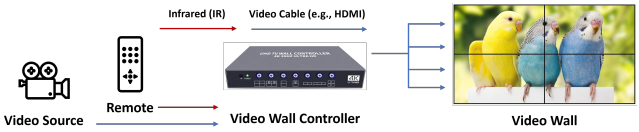


Figure 1: Architecture of a representative video wall system.

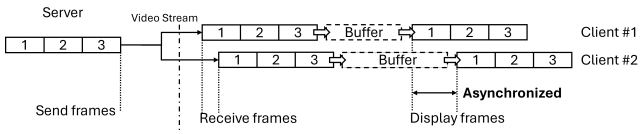
While developing OpenVideoWalls, we identify that maintaining video synchronization among different displays is a key challenge. The network fluctuations or system lags may lead to poor synchronization, which significantly degrades the Quality of Experience (QoE) of the video wall system, no matter how good the other specifications (e.g., resolution and refresh rate) are. Therefore, in this paper, we focus on video synchronization algorithms for heterogeneous displays in the video wall system. To be more specific, we tackle the video synchronization problem from both the server side and the client side. On the server side, we make it a local Network Time Protocol (NTP) [18] server to service the clients so that all the machines can share the same system clock. The client uses the local NTP server to align its time and processes the frames based on their arrival time to achieve intra-frame synchronization among different displays. Furthermore, we conduct comprehensive experiments to evaluate the effectiveness and robustness of our proposed synchronization algorithm using an end-to-end video wall system. To summarize, we make the following contributions in this paper:

(1) We for the first time build an open-source and low-cost solution to enable anyone to build video walls, which supports heterogeneous displays with any layouts.
(2) We for the first time develop an effective video synchronization algorithm to achieve intra-frame synchronization among different displays.
(3) We propose a methodology to evaluate and compare the quality of different video wall systems.
(4) We prototype an experimental system and demonstrate the feasibility of the proposed solution with high quality.

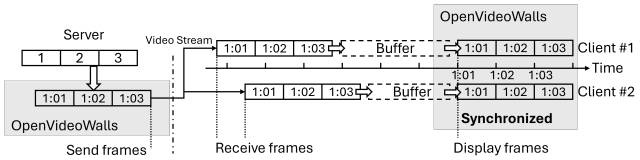
2 Background and Related Work

A video wall is a large display system that consists of multiple displays placed as a grid or custom layout, working together to play a single or multiple videos simultaneously. Video wall technology is becoming more and more popular. Grand-scale immersive and captivating visual experiences are made possible by this technology. However, precise video synchronization is essential to guarantee a smooth and consistent display on every screen. The content presented on several displays may be misaligned, and there may be obvious delays or discrepancies among the screens if synchronization is not handled correctly. To achieve synchronization, the community has proposed several methods. For example, DisplayPort (DP) Daisy Chain [17] has been used to connect the server and several client displays, where all the devices are connected using only one single connection between each pair of devices. In this case, the time differences of the video contents among the displays are extremely small, as they are connected by high-bandwidth DP cables. However, the limitation of this solution is that it only works for small-size video walls like 2x2 or 3x3, due to the limitation of the GPU. Our proposed method has no such limitation as our setup supports multiple servers and multiple streams, which enables a scalable system. Other widely-used methods to synchronize video signals, such as Frame Lock [14] and Reversed Genlock [20], often require expensive and dedicated hardware and thus do not fit our goal of low-cost and open-source. Kato et al. [16] developed Songle Sync for synchronizing events during music playback in a web app, but it only syncs at the application level and is limited to music instead of video, which does not address our need for video signal synchronization. In addition, while protocols like NTP [1], PTP [4], and IRIG [3] are used for time synchronization, they primarily sync the device time instead of the video streams directly.

3 Proposed Approach



(a) Existing streaming system.



(b) Our proposed streaming system.

Figure 2: System overviews of the existing streaming system and our proposed streaming system.

The primary challenge in designing a video wall system is to make every screen display the same frame at the same time. Existing protocols primarily focus on video quality, user experience, etc. on a single display, such as buffering and adaptive bitrate [2], but they lack emphasis on inter-display synchronization. Figure 2(a) shows an overview of the existing streaming system in the multi-display scenario. The *Server* sends out the *Video Stream* for both *Clients* at the same time, marked as *Send frames* in the figure. Due to various impacts such as network latency, *Video Stream* is not guaranteed to be received by the two *Clients* at the same time, as marked by

Receive frames in the figure. Within *Client*, the processing time from receiving *Video Stream* to displaying it is varied due to different factors such as different internal buffer lengths. The different received times and the different processing times lead to asynchronization between different *Clients*. We develop our *OpenVideoWalls* system to solve this problem by correctly managing the frames on the client side and embedding timestamps on the server side. Figure 2(b) shows an overview of our proposed *OpenVideoWalls* system. The *Video Stream* is first handed over to *OpenVideoWalls* to embed the timestamp based on the real-world clock on the *Server*, as marked with the light gray background. The *Video Stream* is then sent from the *Server* for the two *Clients* at the same time (i.e., *Send frames*) with the embedded timestamp. Two *Clients* receive the *Video Stream* at a different time (i.e., *Receive frames*) as expected. Unlike the existing system that displays the received *Video Stream*, *OpenVideoWalls* on *Client* intervenes to ensure that the frame in the *Video Stream* is displayed at the embedded time. Thus, synchronization among different clients is ensured by *OpenVideoWalls*.

3.1 Server-Side Synchronization

We employ a timestamp to indicate when each frame should be shown to the client in order to achieve synchronized displays. Similar functions are included in current video codecs and protocols, such as the presentation timestamp (PTS) [24], which indicates when to display each frame. PTS is designed for scheduling intra-stream frames (e.g., preserving correct frame rate and time intervals between frames); however, it is not intended to address the inter-display synchronization issue.

Using the idea of adding timestamps to frames as in the design of PTS, we have the server attach a Unix timestamp that indicates when each frame should be shown at millisecond precision in lieu of the PTS. This timestamp should ideally be synchronized by the infrastructure among all clients. Client-side frames would be delayed by the network transmission time if the server added the current timestamp while processing the frame. For this reason, as $timestamp = time_{current} + time_{offset}$, the timestamp should be set to a future time relative to the current time when the frame is processed. The server now implicitly controls the buffering rather than the client by setting various offsets ahead of the current time, as in $time_{buffer} = time_{offset} - time_{network}$. Design-wise, the offset should be one to three times the longest possible round-trip time (RTT) with clients, contingent upon how timely the content is. Because the server and the clients synchronize using the Unix timestamp as their foundation, their system clock must be synchronized. To synchronize the clocks, the Network Time Protocol (NTP) [18] is used, and the server serves as an NTP server, giving all clients on the local network the same time to improve the performance.

3.2 Client-Side Synchronization

To achieve synchronization with other clients, the client processes timestamped frames received from the server to display at a predetermined time. The timestamp inserted in the received frame and the client's current time allow the client to classify the frames into three groups:

- *Early*: The timestamp indicates a future time relative to the client's current time.

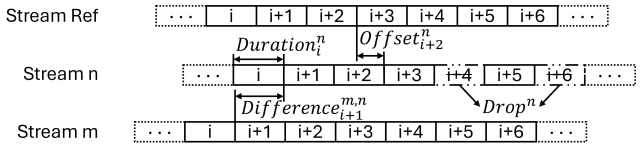


Figure 3: A schematic representation of the four metrics in a two-display system with the virtual reference stream.

- *On-Time*: The timestamp matches the current time or is slightly behind within a tolerable period
- *Late*: The timestamp is beyond the tolerable period behind the current time.

Early frames are expected when the client starts receiving frames from the server. The $time_{buffer}$ for which these frames are retained is specified in Section 3.1. As the first frame in the buffer becomes *On-Time*, the client blocks the buffer to align all of the frames in the buffer to be *On-Time*. *On-Time* frames are immediately shown without any waiting. *Late* frames are dropped to let the client keep up with the server and other clients when they are caused by system slowdowns or unforeseen video decoding delays. In reality, system lag or inaccurate sleep functions would cause some frames classified as *Late* to just slightly deviate from *On-Time*. We further define a tolerance period for *On-Time* frames, based on which any timestamp between the client's current time and the client's current time plus the tolerance is *On-Time*.

3.3 Metrics

As previously mentioned, video wall systems are widely deployed without a proper methodology for evaluation and comparison. Existing video evaluation methods and metrics focus primarily on the performance of a single stream from a content perspective, such as PSNR. Therefore, we propose the following metrics to specifically evaluate video wall systems, as illustrated in Figure 3.

- Dropped Frames $Drop^n$: The quantity of frames omitted during the playback of stream n. To signify the number of frames omitted within a temporal interval from t_i^n to t_j^n, the notation $Drop_{i,j}^n$ is employed.
- Frame Duration $Duration_i^n = t_{i+1}^n - t_i^n$: The time a frame i is displayed.
- Stream Difference $Difference_i^{m,n} = t_i^m - t_i^n$: The temporal disparity between two streams, m and n, during the presentation of the identical frame i.
- Frame Offset $Offset_i^n = t_i^{ref} - t_i^n$: The temporal discrepancy between the scheduled display time of a frame and its actual presentation time.

In the aforementioned metrics, let t_i^n represent the i^{th} frame displayed on stream n, where stream n corresponds to video stream n displayed on screen n. For clarity and simplicity during the comparison, we define a virtual reference stream, denoted as ref, to which all displays should ideally be synchronized. The ref stream functions as a virtual reference stream, allowing all streams and displays to align with it, thereby providing a unified basis for comparison. Based on this virtual reference stream, we further define the metric $Offset_i^n$.

Considering that frames may be dropped during playback for various reasons, such as network impact, video processing capacity limitations, or synchronization requirements, we first propose the metric *Dropped Frames*: $Drop^n$, which counts the number of frames not displayed in the stream. We further define the metric $Drop^n$ as the sum of the dropped frames during the playback of stream n. Ideally, this metric should be zero; practically, the fewer frames dropped, the better. This metric is monotonic and accumulating during playback. To evaluate the system, we define the average of the sum of dropped frames in each stream as the overall $Drop$ metric.

To further evaluate frames shown during playback, we first focus on whether frames are evenly distributed and define the metric *Frame Duration*, $Duration_i^n$, to identify uneven display patterns. For instance, in a 30 frames-per-second video, if 29 frames are displayed in the first 1/30 of a second and the last frame in the final 1/30 of a second, the frames are unevenly distributed. As defined above, we set $Duration_i^n = 0$ if frame i is dropped. Since this metric is a per-frame measure, it should ideally reflect a consistent frame interval (e.g., $Duration_i^n = 33.33ms$ in our experimental setup), with greater stability being preferable. Thus, we define the average of the standard deviation of $Duration$ as the overall $Duration$ metric of the system to represent the stability, where a lower value indicates higher stability.

To evaluate the synchronization between displays in multi-display scenarios, we propose the metric *Stream Difference*, $Difference_i^{m,n}$, to quantify the time differences for identical frames on two displays. We exclude dropped frames from this metric, as they cannot be compared. Ideally, an identical frame should be displayed simultaneously on all displays. This metric is better when it is smaller, with zero representing perfect synchronization across different streams. To evaluate the system, we define the overall $Difference$ metric as the maximum average frame-level $Difference$ observed among all possible stream-pair combinations.

To enhance our understanding and analysis of synchronization from a per-stream perspective, we introduced the concept of the virtual reference stream *ref*. This *ref* stream exhibits the ideal frame duration for each frame and theoretically begins simultaneously with all other streams. Given that *ref* is a virtual stream that can be initiated at any arbitrary time, it is acceptable for the offset between a stream and *ref* to be constant, provided that all streams demonstrate the same or similar offset values. This indicates that they are either ahead of or behind the *ref* stream by the same duration. To quantify the overall offset in the system, we define the metric $Offset$ as the difference between the maximum and minimum average offsets across all streams. A smaller value for this metric indicates better synchronization.

4 System Implementation

OpenVideoWalls consists of three key components in the system implementation as illustrated in Figure 4: *Server*, *Relay*, and *Client*, which are discussed in detail as follows.

4.1 Server

The *Server* is in charge of creating the content to be shown on multiple displays and sending the relevant portions to the stream

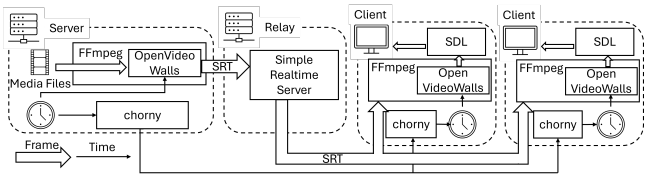


Figure 4: Schematic diagram of the system implementation, which includes a *Server*, a *Relay*, and two *Clients*.

for the client to receive and display. Video processing tasks are handled by the *Server* using the FFmpeg [23] library. For content coming from local files, network streams, and outputs from other programs like OBS [10] and vMix [9], FFmpeg creates a canvas that matches the total size of all displays (e.g. 3840x1080 for two side-by-side 1920x1080 displays). The arrangement of the displays then divides this canvas into sections for output into different streams. The streams are sent over the SRT [5] protocol to *Relay*, encoded in H.264 [24], with resolution and bitrate adjusted to the profile of the intended display.

Every frame has a timestamp attached by FFmpeg to achieve synchronization, as discussed in Section 3.1. The timestamp can be carried in *Supplemental Enhancement Information* (SEI), a feature of the H.264 encoding used for client streams. The bitstream filter h264_metadata in FFmpeg is modified to recognize a particular UUID input and, when the filter processes the frame, replace the corresponding SEI content with the timestamp. The server uses only I-frames to make decoding easier and avoid problems with out-of-order frames and frame dependency, enabling the client to decode each frame as soon as it is received and drop some of them safely if needed.

4.2 Relay

We adopt an open-source streaming server *Simple Realtime Server* [12] (SRS) to relay the SRT stream from the *Server* to the *Client* in order to manage all SRT connections and avoid unintentional disconnections due to problems like timeout. The *Client* can start with SRS even before the *Server* is up. Moreover, it enables several *Server*s to stream to *Client*s since the *Client*s just need to connect to the *Relay* address rather than several *Server* addresses.

4.3 Client

The role of the *Client* is to receive streams from the *Server* and ensure that the frames are shown at the right time according to the embedded timestamps. For display tasks, the *Client* uses the SDL [11] and FFmpeg libraries. The SDL library outputs the decoded frame to the display while the FFmpeg library receives and decodes the stream from the *Server* in two concurrent processes on the *Client*. No B-frame is used in the stream, thus a queue between the two processes stores the decoded raw frames in display order.

The *Client* manages frames in the queue depending on the current time and the timestamp embedded in the frame's SEI in order to achieve synchronization, as discussed in Section 3.2. The *Client* categorizes the frames into three groups: 1) *Early*, *On-Time*, and *Late* as discussed in Section 3.2 When an *Early* frame is encountered, the *Client* sleeps until the timestamp and current time match,

at which point it shows as an *On-Time* frame. *On-Time* frames are shown immediately after being received. *Late* frames are discarded to keep in synchronization with *Server*.

4.4 Time Synchronization

We adopt an open-source NTP implementation *chrony* [6] for time synchronization. *chrony* offers accurate time synchronization under various conditions and includes tools for easy operation and monitoring, which helps achieve clock synchronization between all users and clients. Before starting the experiments, we set up the client systems to use the server as the clock source and provide sufficient time for the systems to synchronize. *chrony* is run continuously to keep the clocks synchronized.

5 Evaluation

5.1 Experimental Setup

To demonstrate the feasibility and the high quality of *OpenVideoWalls*, we run the system in an empirical setup. We use an Ubuntu server with 32GB of memory and 20 vCPUs for the server and to host the SRT Relay. In addition, we use two Raspberry Pi 4 IO boards with 2GB of memory and 4 cores and a Raspberry Pi 5 with 8GB of memory and 4 cores to act as the clients. The clients communicate with the server via the wireless network.

The three clients are connected to three separate display monitors of 1920x1080 resolution, which we refer to as *Monitor #1*, *Monitor #2* and *Monitor #3*. Given that the monitors are positioned horizontally next to each other, we use a video of 5760x1080 resolution and 30 frames per second (FPS) in frame rate to be split over the three monitors as demonstrated in Figure 5(b) for a period of 120 seconds with 1 second tolerance setting on the client. Note that the resolution and display count used for evaluation do not fully represent our system's capacity. Our system can handle inputs of any resolution and adapt them using FFmpeg. Also, with the broadcast-level SRT protocol that does not increase the server load with more clients, our system can scale to a larger number of clients. We report the results for one configuration for clarity.

We also deploy a baseline system for comparison, which uses the same configuration but without the synchronization mechanism, as demonstrated in Figure 5(a). The baseline system differs from our proposed system mostly in that the *Server* lets the *Client* manage the buffering by not attaching timestamps to frames. The FFmpeg toolkit's ffplay program is used by the baseline system's *Client*. We modified the original ffplay to output the PTS and the actual display time for every frame in order to collect results from the baseline system.

As illustrated in Figure 5, the baseline system display in Figure 5(a) clearly exhibits asynchronization. For instance, the middle display shows two complete openings of the belfry, while the right display only shows the very right part of the right-hand-side opening, causing visual tearing. Additionally, there is a noticeable tear at the bottom where the mountain shape meets between the middle and left displays. The tearing caused by screen asynchronization is evident in static video screenshots and significantly impacts user experience in dynamic videos. In contrast, our *OpenVideoWalls* system, shown in Figure 5(b), displays minimal screen tearing due to the synchronization method that will be evaluated in the following sections, reducing frame difference to less than one frame on average across different displays.

(a) Baseline system using FFmpeg. Observable tearing on the building's opening and the mountain shape where cross two displays.

(b) Our proposed *OpenVideoWalls* system. No obvious tearing across the displays.

Figure 5: The demonstration of multi-display video wall systems deployment using baseline method and our proposed method and the tearing issue when the content across different displays in the baseline system.

5.2 Stream Difference

To better evaluate the synchronization across displays, we measure the *Stream Difference* metrics for both systems, as illustrated in Figure 6. Each point in this figure represents the time difference (in milliseconds) for an identical frame displayed across two streams. The baseline system reports the average time differences between each pair of streams as 862.10, 511.16, and 350.98 milliseconds. In contrast, the *OpenVideoWalls* reported differences of 8.36, 28.33, and 20.70 milliseconds for the same stream pairs. Consequently, the overall *Stream Difference* metrics for the two systems are: $Difference_{Baseline}$ = 862.10 ms and $Difference_{OpenVideoWalls}$ = 28.33 ms.

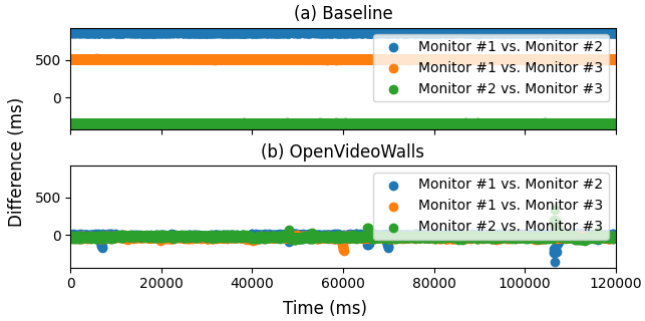


Figure 6: *Stream Difference* for the baseline system (a) and the *OpenVideoWalls* system (b), where each point represents the time difference of displayed time for an identical frame in two streams.

The results demonstrate that our proposed *OpenVideoWalls* exhibits a maximum average difference of 28.33 ms, whereas the

baseline system maintains a nearly constant difference of 862.10 ms, as illustrated in Figure 6. Our *OpenVideoWalls* effectively controls out-of-sync frames to within a one-frame margin in most of the time and actively aligns these frames to minimize differences. In contrast, the baseline system lacks this ability, resulting in a persistent difference that remains unmitigated throughout playback.

5.3 Frame Offset

As illustrated in Figure 7, we further evaluate each stream against the ref stream introduced in Section 3.3. In the baseline system, the average offsets for Monitors #1, #2, and #3 were 863.23, 1.13, and 352.05 milliseconds, respectively. The *OpenVideoWalls* reported the corresponding offsets of 7.48, 0.14, and 20.02 milliseconds. Consequently, the overall *Frame Offset* metrics are $Offset_{Baseline}$ = 862.10 ms and $Offset_{OpenVideoWalls}$ = 19.88 ms.

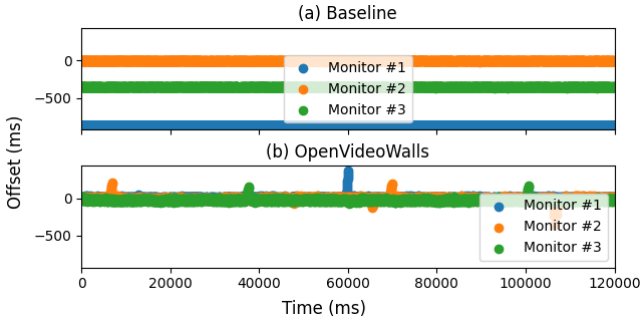


Figure 7: *Frame Offset* for the baseline system (a) and the *OpenVideoWalls* system (b), where each point represents the time difference of displayed time for an identical frame with the *ref* stream.

The *Frame Offset* metric demonstrates that *OpenVideoWalls* outperforms the baseline system in synchronization. Furthermore, as illustrated in Figure 7, this metric indicates that *OpenVideoWalls* does not introduce accumulative delay during playback. It also suggests that the frame-dropping mechanism discussed in Section 5.4 functions effectively for frames that arrive late.

5.4 Dropped Frames

As described in Section 3.3, we quantify frame drops in both the baseline and the *OpenVideoWalls* systems by identifying frames with zero duration in the log files. In the baseline system, Monitors #1,#2, and #3 report 3, 8, and 7 dropped frames, respectively. The *OpenVideoWalls* system causes 3, 46, and 38 dropped frames for the same monitors. Consequently, the overall *Dropped Frames* metric for the two systems are: $Drop_{Baseline}$ = 6 and $Drop_{OpenVideoWalls}$ = 29.

The overall metric identifies that our proposed approach introduces additional frame dropping that could degrade the user experience compared to the baseline system as an overhead. Based on data collected from the 120-second video at 30 FPS, the average frame drop ratio is 0.17% for the baseline system and 0.81% for the *OpenVideoWalls* system. Both ratios are considered acceptable in common scenarios, considering the case the actual frame rate of our proposed system (29.76 FPS) displayed on a 30 FPS screen vs. a NTSC format video (29.97 FPS) displayed on a 30 FPS screen.

5.5 Frame Duration

We measure the *Frame Duration* metric for displayed frames to further evaluate the impact of frame dropping, as illustrated in Figure 8. Each point represents a frame's duration (y-axis, in milliseconds) at a specific time (x-axis, in milliseconds). The baseline system reports standard deviations of frame duration across all frames as 0.72, 2.16, and 1.90 milliseconds for Monitors #1,#2, and #3, respectively. The *OpenVideoWalls* reports corresponding standard deviations of 20.06, 17.62, and 17.06 milliseconds. Consequently, the overall *Frame Duration* metrics for the two systems are $Duration_{Baseline}$ = 1.59 ms and $Duration_{OpenVideoWalls}$ = 18.25 ms.

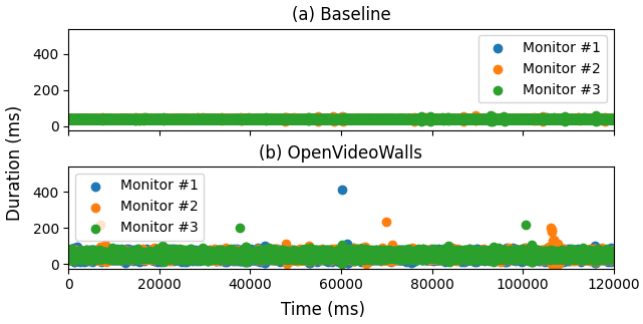


Figure 8: *Frame Duration* for the baseline system (a) and the *OpenVideoWalls* system (b), where each point represents a frame's duration.

Although both systems achieve *Frame Duration* less than one frame interval (33.33 milliseconds), the *OpenVideoWalls* introduces a larger standard deviation as an overhead, indicating a broader distribution of frame duration. This drawback primarily results from our attempt to synchronize all streams during playback. Unlike the baseline system, which displays frames with relatively consistent duration, *OpenVideoWalls* allocates frames within a range where all streams can display simultaneously, as discussed in Section 3.2. This approach leads to a broader duration distribution as a trade-off for improved synchronization.

6 Conclusion and Future Work

In this paper, we developed an open-source and low-cost video wall system, namely *OpenVideoWalls*, which allows anyone to build a low latency and high QoE video wall with multiple displays. This is our first attempt to build a video wall system contributing to display recycling, and we believe more improvements to the system can be conducted in the future work, such as further reducing the *Frame Duration*, a fully automatic configuration of monitors and clients, and a more user-friendly interface. To motivate further research and development in the community, we have released the source code of *OpenVideoWalls* at https://github.com/hwsel/multi-screen.

Acknowledgments

This work was supported in part by the National Science Foundation under awards 2235364, 2329704, and 2229873 and by the gift donation from Adobe Inc.

References

[1] 2010. Network Time Protocol Version 4: Protocol and Algorithms Specification. https://datatracker.ietf.org/doc/html/rfc5905

[2] 2012. P.1201: Parametric Non-Intrusive Assessment of Audiovisual Media Streaming Quality. https://www.itu.int/rec/T-REC-P.1201/en

[3] 2016. IRIG Serial Time Code Formats. https://apps.dtic.mil/sti/tr/pdf/ADA640534.pdf.

[4] 2019. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. https://doi.org/10.1109/IEEESTD.2020.9120376

[5] 2023. *Secure Reliable Transport (SRT) Protocol.* https://github.com/Haivision/srt.

[6] 2024. Chrony – Introduction. https://chrony-project.org/.

[7] 2024. Digital Screens & Billboards, Times Square. https://www.timessquarenyc.org/do-business/promote-partner/advertising-sponsorships/digital-screens-billboards.

[8] 2024. The global E-waste Monitor 2024 – Electronic Waste Rising Five Times Faster than Documented E-waste Recycling: United Nation. https://ewastemonitor.info/the-global-e-waste-monitor-2024/.

[9] 2024. Live Video Streaming Software | vMix. https://www.vmix.com/.

[10] 2024. Open Broadcaster Software | OBS. https://obsproject.com/.

[11] 2024. Simple DirectMedia Layer. https://www.libsdl.org.

[12] 2024. Simple Realtime Server. https://github.com/ossrs/srs.

[13] Matthias Baldauf and Peter Fröhlich. 2013. The augmented video wall: multi-user AR interaction with public displays. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*. 3015–3018.

[14] Marcelo P Guimarães, Paulo A Bressan, and Marcelo K Zuffo. 2002. Frame lock synchronization for multiprojection immersive environments based on PC graphics clusters. In *SBC Symposium on Virtual Reality*. 89–96.

[15] Masaki Hayashi, Steven Bachelder, Masayuki Nakajima, and Akihiko Iguchi. 2014. A new virtual museum equipped with automatic video content generator. In *International Conference on Cyberworlds*. 377–383.

[16] Jun Kato, Masa Ogata, Takahiro Inoue, and Masataka Goto. 2018. Songle Sync: A large-scale web-based platform for controlling various devices in synchronization with music. In *ACM international conference on Multimedia (MM)*. 1697–1705.

[17] Alan Kobayashi. 2010. DisplayPortTM Ver. 1.2 Overview. In *DisplayPort Developer Conference*.

[18] David L Mills. 1991. Internet time synchronization: the network time protocol. *IEEE Transactions on communications* 39, 10 (1991), 1482–1493.

[19] Olga Mirgorodskaya, Olesya Ivanchenko, and Narine Dadayan. 2020. Using digital signage technologies in retail marketing activities. In *International Scientific Conference-Digital Transformation on Manufacturing, Infrastructure and Service*. 1–7.

[20] Jochen Miroll, Alexander Löffler, Julian Metzger, Philipp Slusallek, and Thorsten Herfet. 2012. Reverse genlock for synchronous tiled display walls with Smart Internet Displays. In *IEEE International Conference on Consumer Electronics-Berlin (ICCE-Berlin)*. 236–240.

[21] Nirnimesh Nirnimesh, Pawan Harish, and PJ Narayanan. 2007. Garuda: A scalable tiled display wall using commodity PCs. *IEEE Transactions on Visualization and Computer Graphics* 13, 5 (2007), 864–877.

[22] Vasiliki Savvilotidou, John N Hahladakis, and Evangelos Gidarakos. 2014. Determination of toxic metals in discarded Liquid Crystal Displays (LCDs). *Resources, Conservation and Recycling* 92 (2014), 108–115.

[23] Suramya Tomar. 2006. Converting video formats with FFmpeg. *Linux journal* 2006, 146 (2006), 10.

[24] Thomas Wiegand, Gary J Sullivan, Gisle Bjontegaard, and Ajay Luthra. 2003. Overview of the H. 264/AVC video coding standard. *IEEE Transactions on circuits and systems for video technology* 13, 7 (2003), 560–576.

[25] Aya Yoshida, Atsushi Terazono, Florencio C Ballesteros Jr, Duc-Quang Nguyen, Sunandar Sukandar, Michikazu Kojima, and Shozo Sakata. 2016. E-waste recycling processes in Indonesia, the Philippines, and Vietnam: A case study of cathode ray tube TVs and monitors. *Resources, Conservation and Recycling* 106 (2016), 48–58.