Extensible Immersive Virtual Environments for Large Tiled Video Walls

Lorenz Cuno Klopfenstein[1], Brendan D. Paolini[1], Gioele Luchetti[2] and Alessandro Bogliolo[1]
[1]*DiSBeF, Università di Urbino "Carlo Bo", Urbino, Italy*
[2]*DII, Università Politecnica delle Marche, Ancona, Italy*

Keywords: SAGE, IVE, Immersive Environment, Immersive Scenario, Virtual Environment.

Abstract: The intent of the work is to present an *Immersive Virtual Environment* (IVE) as a new abstracted management layer on top of the *Scalable Adaptive Graphics Environment* (SAGE) system, allowing the simplified management and linear scaling of multiple SAGE-driven video walls, and the creation and control of simple interactive scenarios. The framework exposes evolved *Application Programming Interfaces* (APIs) that are detached from the underlying system and can be used by mobile clients as well. Primitives offered to developers and content creators allow the definition of immersive cinematographic experiences using basic commands, which are synchronized on the whole IVE environment. A complete implementation of the system is described and then evaluated with the specific case of one physical installation.

1 INTRODUCTION

High resolution immersive displays are becoming increasingly common, for large scale interactive entertainment, for "situation-room" displays, and for massive data visualization, where ultra-large images with several millions of pixels are visualized on a single coherent surface, including walls or monitor arrays.

Typical use-case scenarios include local and remote groups of people working together on distributed and heterogeneous data, obtaining the ability to control, manipulate, share, and visualize information. Collaborative work on ultra-high-resolution displays with massive data has been found to be crucial for several kinds of research and analysis (Renambot et al., 2004).

As envisioned in different existing projects (Smarr et al., 2003; Klosowski et al., 2002), large-scale visualizations can neither be adapted to traditional desktop displays, nor to standard projection techniques. Instead they usually require special setups with a combination of very specific software and hardware.

For instance, when large scale visualizations are not feasible with a single wide-*Field Of View* (FOV) projector, with very expensive optics and hardware (Zobel Jr et al., 1998), they are achieved using tiled displays. That is, splitting the high-resolution image into multiple separate regions, on different displays. This has been successfully adopted by using non-overlapping projections (DeFanti et al., 2011), side-by-side projections with precisely aligned overlaps, or tiled LCD panels (Ni et al., 2006). Quite often, these installations are very expensive and difficult to replicate.

A secondary issue of such large-scale visualization systems, which can aggregate inputs from different sources and heterogeneous video data, is that applications have to be re-designed or re-adapted in order to be effectively used on that particular graphical environment, and they rarely take advantage of the full graphical capabilities of a dedicated large visualization system.

Most solutions in this field do not provide unified control of both input and output of the system. The lack of higher abstraction interfaces for input and output also entails a higher difficulty adapting the setup to specific needs, which usually necessitates very challenging integration work.

Some existing solutions, like LOTUS (Cho et al., 2012), do indeed support predefined scenarios and high-level input interfaces, but do not support multiple visualizations and dynamic reconfiguration.

The solution proposed by this paper is tailored for multi-room entertainment systems or immersive interactive housing (i.e. multiple independent large-scale visualizations that require central supervision). The system is able to manage multiple concurrent visualizations, which are part of one overarching scenario, and to reconfigure them dynamically.

Klopfenstein, L., Paolini, B., Luchetti, G. and Bogliolo, A.
Extensible Immersive Virtual Environments for Large Tiled Video Walls.
DOI: 10.5220/0005727803210328
In *Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2016) - Volume 1: GRAPP*, pages 323-330
ISBN: 978-989-758-175-5

2 SAGE

The *Scalable Adaptive Graphics Environment* (SAGE) is a cross-platform open-source middleware, that was developed as a flexible graphics streaming system. It enables users to compose multiple heterogeneous visualization applications seamlessly and in real-time, on a very large display. Its decoupled architecture allows rendering applications to take advantage of the processing power of the platform they were developed for, without having to be constrained or re-designed for SAGE's graphic environment. At the same time, compatibility with a large number of data sources in a variety of resolutions and formats is ensured (Renambot et al., 2004).

The primary aim of this software system is to let local or distributed groups of users access, display and collaborate with ease on large datasets, in a variety of resolutions and formats, which require a big-scale visualization. Data may be generated by high-performance storage and rendering clusters and streamed over a high-bandwidth network in order to be displayed on collaborative graphical end-points, which may range widely in both surface area, resolution and type of display (for instance a single high resolution screen, a large tiled display or an array of projectors). End-points provide means to efficiently visualize data from multiple sources and to interact with it.

SAGE has been put in use successfully in several installations, including the *CAVE* (DeFanti et al., 2011) and *LambdaVision* at the *Electronic Visualization Laboratory* at the University of Illinois at Chicago (EVL-UCI) (Renambot et al., 2005).

2.1 Architecture

SAGE is designed following a flexible and scalable architecture, allowing multiple applications and video sources to be streamed to a variety of displays, without requiring any special hardware.

The system supports a single "scalable virtual frame buffer" (a rectangular pixel matrix of almost arbitrary size), which is tiled logically according to the system's configuration, matching the actual system's topology. One or more tiles, which make up a contiguous screen area, are managed by a *SAGE Receiver*. It can receive a certain number of inbound pixel streams, compose them and then push them to the screens for display, as shown in Figure 1.

The *Free Space Manager* (FSM) acts as a window manager for SAGE and collects and maintains the information about receivers and other components. The FSM orchestrates the configuration of the system through a message passing protocol, directly communicating with other components and dinamically reconfiguring the system when needed (Jeong et al., 2006).

Pixel streams are generated by *SAGE Applications*. These are generic multi-purpose programs that may run on any server of the system and make use of the *SAGE Application Interface Library* (SAIL) to provide image data to the *SAGE Receivers*.

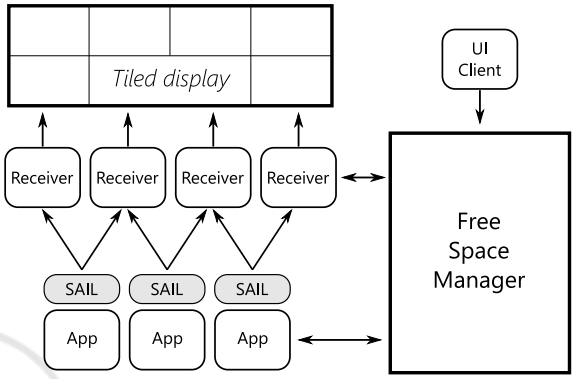

Figure 1: SAGE architecture overview.

This distributed architecture allows the receivers to be logically separated from the applications actually generating pixel data. Applications can run on the same device of the target receiver, on a different machine, or even on a remote machine connected through a wide-area network.

Each single application thus contributes to the tiled frame buffer, possibly pushing data through a high-bandwidth network that bridges the computers composing the SAGE system.

The FSM controls the pixel stream configuration between applications and receivers: each application is assigned to a specific region of the frame buffer to draw to, which may change as applications are started, terminated, moved across the frame buffer and scaled in size. The FSM makes sure that each application streams pixel data to the correct receivers and that the pixels are thus drawn to the exact output region on the frame buffer.

2.1.1 Applications and SAIL

The *SAGE Application Interface Library* (SAIL) provides a facility for programs to work as a *SAGE Application*, by interacting with the FSM and by generating pixel data to be shown on the frame buffer.

Data streamed through SAIL is encoded and transmitted as raw pixels, usually in 24-bit RGB format or using the lightweight lossless DXT compression format. Transmission occurs without any additional con-

version and without incurring in the cost and quality penalty of compression. While this also ensures that the protocol has no strong platform dependency, it makes it subject to very high bandwidth requirements.

SAIL represents a very thin layer between application and network, that allows application developers to generate output pixels and transparently stream them. The programming interface is built around a very simple buffer-swapping mechanism: the application generates raw pixel buffers and swaps them out for SAIL to push them to the receivers. If necessary—for instance when a *SAGE Application*'s output region spans across multiple output tiles—SAIL takes care of splitting the pixel data and streaming it to the correct *SAGE Receivers*, according to the system's configuration.

Existing programs, for most platforms and programming languages, can easily be adapted to take advantage of the large visualization capabilities of SAGE thanks to this simplified streaming model. In fact, SAGE by default provides a modified version of *MPlayer* that is capable of working as a *SAGE Application* by using a custom output video module.

2.2 Interoperating with SAGE

The SAGE FSM exposes a developer-facing interface, called "Event Communication Layer", that allows SAGE UI modules and third-party controllers to interact with the system.

Clients can send user messages to control the FSM and the applications managed by it, for instance setting up a new application, moving it from one location of the frame buffer to another or resizing its output region. In return, clients obtain SAGE event messages informing them about the current state of the system and its running applications.

The layer is based on a low-level text message exchange protocol on top of a TCP socket between client and FSM. It supports high rates of message exchanges, but the messages use a basic format and are limited to very simple information (see section 3.3).

2.3 Requirements

Large-scale collaborative visualization environments intrinsically have very high requirements, for both software and hardware.

In a SAGE-based system, when a *SAGE Receiver* and an application using SAIL are not located on the same machine, the amount of video data that must be streamed through the network can be substantial. This puts a very high performance requirement on the network between nodes, which calls for high-grade multiple gigabit network equipment to be used.

Especially when reaching 4K resolutions or higher, transmitting and handling video in real-time puts a very high strain both on processing components and the network. This can be exarcebated by the need to handle different resolutions or encodings, and to synchronize playback (Singh et al., 2004).

Finally, *SAGE Receivers* that individually drive several displays require a custom multi-monitor setup which usually also requires expensive video hardware.

3 IMMERSIVE VIRTUAL ENVIRONMENT

In the following section the *Immersive Virtual Environment* (IVE) system will be presented. The system is built on top of SAGE and presents a more extensible way to coordinate immersive multisensorial scenarios on one or more high resolution displays.

IVE is designed to control a multimedia interactive scenario, taking place on multiple video outputs and a variety of other sensorial actuators, like surround audio systems, odor diffusers, etc.

As described in section 2.1, the SAGE system has no higher abstraction understanding of the lifetime of applications that draw on its frame buffer, nor of how these applications interact with each other and other components. IVE tackles this issue by wrapping one or more SAGE instances and implementing a high level API on top of them, thus enabling any client application to describe, program and control complex multimedia scenarios.

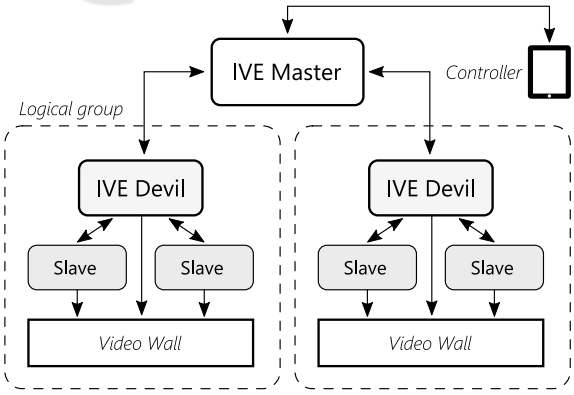

Figure 2: IVE architecture overview.

3.1 Architecture

The *Immersive Virtual Environment* (IVE) system is designed to support multiple independent video walls. IVE installations are split in logical groups, each one of which is bound to a display surface that has no intersections with surfaces of any other group.

IVE is structured in a centralized master/slave topology as depicted in Figure 2: the whole system is managed by a single supervising *IVE Master*, while each independent display region (i.e., a logical group) is controlled by an *IVE Devil* server. On its turn, each *Devil* may control any given number of *IVE Slave* servers, whose activities and interactions are still orchestrated by the master server.

Components of an IVE system are the following:

- *IVE Master*: is a lightweight standalone server implemented in Java, exposing a high-level communication protocol. It manages the system state, controls communication channels to the clients and to groups of slave servers.
- *IVE Devil*: overviews a single logical IVE group, dedicated to an independent display surface. The server receives commands and communicates events back to the *Master*, thus keeping the system's state in sync. A Devil server internally also runs a *SAGE Free Space Manager* instance and controls it through its "Event Communication Layer" (see 2.2), thus controlling the output to the video wall.
- *IVE Slave*: is a passive worker of a logical IVE group. There may be multiple instances inside the same group, each running a *SAGE Receiver* process internally. A Slave also has the ability to run any number of *SAGE Applications*. Local and remote applications can stream video data to their *SAGE Receiver* and thus contribute to the video output.

Generally, each component of IVE owns and controls one or more SAGE components internally. Every IVE logical group contains one *Devil* and any number of *Slave* instances, which on their turn respectively manage the SAGE FSM (configured to work on the logical group's display), the *Receivers* and any *Application* instance.

Which and how many instances of SAGE components are handled by what IVE component is described in Table 1.

On a large system, components will normally be located on multiple machines in order to split the load and to accomodate for physical distance between components: a *Devil* node will control a certain number of *Slaves* installed on separate machines, depending on the requirements of the installation. The total number of machines needed mainly depends on the size of the video wall and the covered surface's geometry.

Table 1: Relationship between IVE and SAGE components.

IVE	**SAGE**
Master	—
Devil	Free Space Manager (1) Receiver (0-1)
Slave	Receiver (1) Application (0+)

The *SAGE Receiver* instance running on an *IVE Slave* node can drive a number of displays (and corresponding tiles on the video wall) that is limited only by its graphical hardware and, possibly, by software limitations. Off-the-shelf consumer hardware and software usually cannot run more than four displays at a time from a single workstation. These constraints aside, the whole output surface of a logical group can be managed by a single *SAGE Receiver* instance.

In fact, on small-scale systems all components can in principle run on the same physical machine. Both the role of the *Devil* and the *Slave* can be taken over by the same node, which will also manage all SAGE components required (the *FSM*, the *Receiver* and all applications generating video data). Moreover, the *IVE Master* itself can be run on the same machine as well, in order to handle communication with client controllers.

3.2 IVE Management

All IVE components, *Master*, *Devil* and *Slave*, are implemented in Java and run as standalone headless processes running on a standard *Java Virtual Machine* (JVM) instance.

Underneath the IVE software layer, *IVE Devil* and *Slave* nodes will manage their respective SAGE processes following the commands issued by the *Master* node. All SAGE components run as standalone processes, started and monitored by the IVE software: this includes the FSM (managing the whole frame buffer directly mapped onto the display surface of that particular IVE group), *SAGE Receivers* (managing tiles of the frame buffer) and all applications generating the output video data.

When the *IVE Master* detects a new node coming online (according to the procedure in section 3.4) it opens a new bidirectional communication using TCP sockets. Additionally, controller clients that manage the IVE installation open a TCP communication channel to the *Master* node.

Communication between the *IVE Master* and other parties occurs through a simple message exchanging protocol. Messages sent by the Master or by a Controller client are called *Commands*, while those sent in the opposite direction are called *Events*, updating the Master or a Controller about a state change occuring at a lower level.

IVE nodes support a variety of commands and events, including:

- Starting or stopping SAGE instances, shutting down the system.
- Starting, moving, resizing, rotating, terminating or setting the transparency of applications. Video players can start playback, pause or seek inside a video file.
- Notifications about a change of application state.

Messages are encoded as simple JSON objects. Each message contains at least a "type" member with a string value, describing the kind of command or event it represents. Additional data is encoded as further members of the message JSON object.

For instance, the command to resize an application running on an *IVE Slave* is formatted as follows on the wire:

```
{
    type: "SAGE_CMD_RESIZE_W",
    appId: <SAGE APP ID>,
    left: 0, right: 100,
    top: 0, bottom: 100,
    isFullscreen: false
}
```

In this case, the command sent by a Controller is forwarded by the *IVE Master* to the *IVE Devil* managing the application with a SAGE ID matching the '`appId`' parameter. The *IVE Devil* then manipulates the SAGE FSM to comply with the resize command, as further described in section 3.3.

In addition to commands to start or manipulate applications, the IVE protocol also exposes a rich set of APIs that allow Controller clients to query the state of the system and present a coherent user interface to the end-user. Any device capable of connecting to the network and exchanging JSON messages through TCP can be used to control the system.

By relying on the same Java code base of the IVE servers, an Android application was developed, allowing the user to easily and interactively control the scenario using a tablet.

The message exchange protocol between IVE nodes is very lightweight, requiring only the transmission of simple text messages, thus IVE nodes do not need to be connected by a high-bandwidth network.

3.3 Interoperating with SAGE

A SAGE installation is composed by several independent processes, that cannot easily be managed through a single interface.

As described before, SAGE provides a simple SAIL interface for applications to stream video data to the output display (through a *SAGE Receiver*), but this interface provides no coherent way to manage the application itself or to interact with it. In fact, other extensions to SAGE have been proposed in order to overcome this limitation and to let users control applications (Fujiwara et al., 2011).

The default "Event Communication Layer" provided by the SAGE FSM exposes a simple text-based interface to broadcast updates, in the form of strings separated by newlines, and taking commands in a similar form. The layer allows high communication rates with low overhead. However the barebones format of the messaging protocol make the task of keeping track of the system's status burdensome.

SAGE messages have 4 fixed-size data fields and 1 payload field, all separated by null characters, following the layout seen in Table 2.

Table 2: Layout of a SAGE message.

Distance	8 bytes
Code	8 bytes
App code	8 bytes
Size	8 bytes
Payload	—

The *Distance* and *App code* parameters appear to be generally unused. Message *Codes* are defined by the SAGE documentation: commands are in the 1000–1100 range, while event codes start from 40000. *Payloads* are expressed as a string of parameters separated by ASCII spaces (Jeong et al., 2005).

For instance, the same resizing command generated by the *IVE Master* seen before will generate a SAGE command on the *IVE Devil* running the application as seen in Figure 3. The '`1004`' value represents the command code for resizing. Inside the payload, the '`ID`' value will be replaced by the actual SAGE application ID of the target application.

Application IDs are unique for each SAGE Application in the context of a single FSM server. When a new application process is spawned by a command by the *Master*, the *IVE Devil* registers the application to its FSM instance using a SAGE command with code '`1001`'.

In order to keep the association between running processes and application registrations up to date, all *IVE Devil* instances also listen to application updates

```
Distance                    Code
0 0 0 0 0 0 0 0 NUL 0 0 0 0 1 0 0 4 NUL
App Code                    Size
0 0 0 0 0 0 0 0 NUL 0 0 0 0 0 0 1 4 NUL
Payload
ID 0 100 100 0
```

Figure 3: Sample SAGE message of a resize command.

from the FSM process, which appear as SAGE messages with code '40001'. Any update to the application's state are surfaced through these event messages and intercepted by the IVE process. These updates contain information such as the unique ID assigned to a newly registered application and its current position, size and z-axis ordering.

At the same time, updates are also forwarded to the *IVE Master* server, in the form of an IVE event that contains a reference to the application, a reference to the *IVE Devil*, and the ID generated by the FSM instance owning the application (i.e., controlling the frame buffer the application is contributing to and where it is registered at). The *Master* server keeps an updated map of all running applications and their metadata, in order to uniquely identify running processes and to track their activity.

How SAGE and IVE components exchange messages, commands and events is represented in Figure 4.

The server attempts to keep a coherent and complete vision of the entire system, which goes beyond the limited vision that a SAGE server has about its environment and the Application running therein. For instance, the *IVE Master* can keep track of whether a video player application is playing, paused or stopped—based on its last control interaction—and is able to provide a full snapshot of the system's state through its APIs. Commands, events and status queries exposed by IVE enable a much more powerful and rich message exchange between the system and its client controllers.

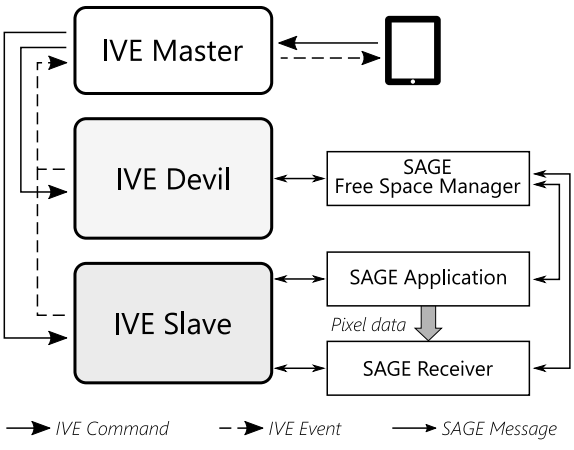

Figure 4: Message routing.

3.4 Auto Discovery

Being intended to work on dedicated installations, SAGE mostly relies on static configuration files for its setup. Instead, IVE is designed to work in an environment where subsets of the system may not be available or may be intentionally turned off. Moreover, all components of an IVE installation are meant to be located on the same network. Thus, IVE can rely on an auto discovery process in order to detect the available components as they come online.

System start-up works as follows:

1. *IVE Master* and *Devil* processes start as soon as the boot phase of the systems where they respectively reside is completed.
2. *IVE Devil* nodes announce their presence with periodical UDP broadcast messages on the network.
3. The *IVE Master* will detect newly started *Devils*, register them and initiate a communication channel. *Devils* are setup by the *Master* according to the configuration defined by the user during the initial installation.
4. *IVE Devils* start their SAGE processes, including the FSM and the Receiver. These SAGE configurations are highly tailored to their hardware setup and depend on preset local configuration files.
5. The *Master* node will then load the preconfigured scenario, starting applications by sending commands to the *Slaves*, and begin receiving events from them in order to maintain track of the system's state.

3.5 Load Balancing

A *SAGE Application* can be drawn on any region of the video wall, sometimes overlapping two or more tiles which are controlled by different *SAGE Receivers*. When the application and the receiver are located on different physical machines, some part of the video data generated by the application must be transferred through the network in order to reach the video wall. Since SAGE mostly transfers raw video data, this may put a very high strain on the network, quickly reaching gigabit-per-second bandwidths (Singh et al., 2004).

IVE is designed to keep a high level overview of where applications run and where they are displayed.

Thus, the *IVE Master* node takes into account the target region where an application will be shown and, when starting a new *SAGE Application* process to generate video data, will spawn the process on the nearest machine to the *Receiver* (ideally on the very same machine).

As seen in Figure 5, when an application is running on the same *IVE Slave* as its receiver, all data transfer through the network can be avoided. When a part of the target region, albeit small, overlaps on another tile of the video wall, the video data must be streamed from the application to the corresponding receiver, which requires a high-bandwidth data transfer.

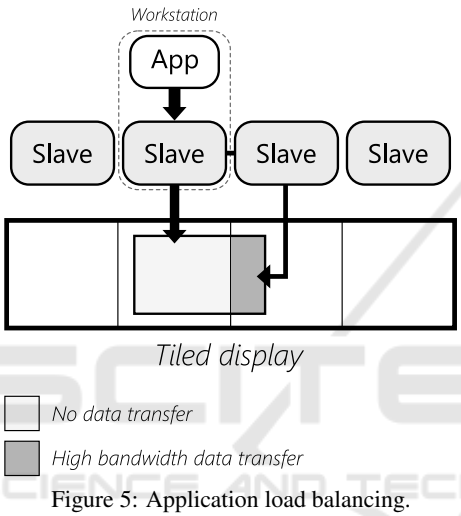

Figure 5: Application load balancing.

At the moment, the *IVE Master* only takes into account the initial starting region where an application will be drawn when selecting a machine to spawn the process. If an application is moved or scaled, its draw region may move to another tile. In this case, relocating an application process from one machine to another is possible for simple, static applications (e.g., image viewers), but still poses a challenging problem for applications like videos or rendered outputs, which would need to transfer their state and resume seamlessly on another machine.

4 HARDWARE SETUP

A prototypal installation of the IVE system has been realized in a dedicated room for demonstration and testing purposes, shown in Figure 6. The main wall of the room measures about 11 × 3 m, while secondary side walls measure 2 × 3 m.

The setup is intended to entirely cover the main surface and the lateral walls of the room with a single complex visualization. To this purpose, 8 BenQ LW61ST projectors where used for the main surface and 2 identical projectors for each side wall, totaling 12 projectors to cover the entire surface. The overall projected frame buffer accounts for 7680 × 1440 total pixels (roughly 11 mega pixels).

Figure 6: IVE installation running an immersive scenario.

The surface is split in three logical regions, each managed by a single *IVE Slave* node. The workstation managing the main screen also runs the *IVE Master* and a *Slave* server at the same time. (The adopted setup makes IVE run as a standalone system, as a single logical group, however the *IVE Master* may in fact be located outside the room or even remotely, while also managing other *IVE Devils* at the same time.)

All IVE nodes are connected by a high-performance 10 gigabit network[1], which supports high-bitrate raw frame buffer data passing from one server to another. The network makes use of one Netgear Prosafe XS708E gigabit ethernet switch and workstations are equipped with Intel X540-T1 network cards.

The central workstation running the *Master* and the main *Devil* instance has the more onerous workload, having to drive 8 projectors while also controlling the other two machines. The workstation is equipped with two NVIDIA Quadro K5000 video cards (each sporting 2 DVI and 2 Displayport video outputs) and an NVIDIA Quadro Sync card. The two secondary computers driving the 4 projectors on the side rooms are equipped with single NVIDIA GTX670 video cards (providing 2 video outputs each). Both kinds of workstation are also equipped with Intel i7 3770 processors, SSDs and respectively 8 and 4 GB of RAM.

[1]This kind of high-bandwidth network was found to be necessary for the playback of 4K videos. For lower resolution video sources, a 1 gigabit network would suffice.

Each machine runs Ubuntu 12.04 LTS, using the Compiz desktop manager and the *LightTwist* plug-in for projection deformation correction and alignment.

5 CONCLUSIONS

The proposed IVE system provides an extensible management solution for complex video installations, composed of one or more large-scale video walls driven by SAGE.

The IVE system is capable of managing multiple SAGE instances, hiding the complexity inherent in managing and interacting with SAGE processes and applications through a coherent front-end that keeps track of the underlying system's state. All of IVE's features are accessible through high-level APIs that can be used by any software client, including mobile applications.

IVE also provides several extension points, possibly including interfaces for other systems and protocols (including KNX for domotic devices, lighting, odor diffusers, etc). Interactive or passive scenarios can be implemented through the programmable APIs.

5.1 Future Work

In the near future, the adoption of multimedia installations made possible by IVE will expand beyond the usage in academic institutions and art installations, eventually reaching offices and private homes. Such large-scale visualizations will not only be limited to collaborative work, but also include immersive entertainment, multisensorial experiences and even more passive uses, like digital wallpapers.

Such scenarios shall be explored, providing new use-cases for the adoption of IVE other than large data visualization and entertainment. In fact, the prototype used to test IVE has been adapted and will be tested in a interactive multisensorial housing installation.

Furthermore, since IVE relies on a large number of distributed components, the possibility of running the whole system or parts of it on low-power embedded systems will be explored. In particular, machines running specific applications or the SAGE FSM could be replaced by a cluster of embedded devices that can be efficiently managed by the supervising *IVE Master* server.

REFERENCES

Cho, Y., Kim, M., and Park, K. S. (2012). Lotus: composing a multi-user interactive tiled display virtual environment. *The Visual Computer*, 28(1):99–109.

DeFanti, T. A., Acevedo, D., Ainsworth, R. A., Brown, M. D., Cutchin, S., Dawe, G., Doerr, K.-U., Johnson, A., Knox, C., Kooima, R., et al. (2011). The future of the CAVE. *Central European Journal of Engineering*, 1(1):16–37.

Fujiwara, Y., Ichikawa, K., Takemura, H., et al. (2011). A multi-application controller for SAGE-enabled tiled display wall in wide-area distributed computing environments. *Journal of Information Processing Systems*, 7(4):581–594.

Jeong, B., Jagodic, R., Spale, A., Renambot, L., Aguilera, J., and Goldman, G. (2005). SAGE documentation. https://www.evl.uic.edu/cavern/sage/documentation/SAGEdoc.htm. Accessed: 2015-09-28.

Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A., and Leigh, J. (2006). High-performance dynamic graphics streaming for scalable adaptive graphics environment. In *SC 2006 Conference, Proceedings of the ACM/IEEE*, pages 24–24. IEEE.

Klosowski, J. T., Kirchner, P. D., Valuyeva, J., Abram, G., Morris, C. J., Wolfe, R. H., and Jackman, T. (2002). Deep view: high-resolution reality. *Computer Graphics and Applications, IEEE*, 22(3):12–15.

Ni, T., Schmidt, G. S., Staadt, O. G., Livingston, M., Ball, R., May, R., et al. (2006). A survey of large high-resolution display technologies, techniques, and applications. In *Virtual Reality Conference, 2006*, pages 223–236. IEEE.

Renambot, L., Johnson, A., and Leigh, J. (2005). LambdaVision: Building a 100 megapixel display. In *NSF CISE/CNS Infrastructure Experience Workshop, Champaign, IL*.

Renambot, L., Rao, A., Singh, R., Jeong, B., Krishnaprasad, N., Vishwanath, V., Chandrasekhar, V., Schwarz, N., Spale, A., Zhang, C., et al. (2004). SAGE: the scalable adaptive graphics environment. In *Proceedings of WACE*, volume 9, pages 2004–09. Citeseer.

Singh, R., Jeong, B., Renambot, L., Johnson, A., and Leigh, J. (2004). Teravision: a distributed, scalable, high resolution graphics streaming system. In *Cluster Computing, 2004 IEEE International Conference on*, pages 391–400. IEEE.

Smarr, L. L., Chien, A. A., DeFanti, T., Leigh, J., and Papadopoulos, P. M. (2003). The OptiPuter. *Communications of the ACM*, 46(11):58–67.

Zobel Jr, R. W., Bennett, D. T., Idaszak, R. L., and Kovach, D. (1998). Multi-pieced, portable projection dome and method of assembling the same. US Patent 5,724,775.