ZAPP – A management framework for distributed visualization systems

Georg Tamm[1,2], Alexander Schiewe[2], Jens Krüger[1,2,3]
[1]*Interactive Visualization and Data Analysis Group, DFKI, Saarbrücken, Germany*
[2]*Intel Visual Computing Institute, Saarbrücken, Germany,* [3]*SCI Institute, Salt Lake City, USA*

ABSTRACT

In this work we present the ZAPP, a management framework for distributed visualization systems. The main purpose of ZAPP is to improve the user, administrator, and developer experience in dealing with distributed displays. Even a single user with no administrative knowledge can operate a ZAPP-managed display system. The goal is that practically anyone is capable of operating the display with less than two minutes of training. To achieve this we present a lightweight management framework that links and controls the render workstations, which drive the displays, but also connects to the user's very own mobile devices, such as smartphones or tablets, to enable convenient control over the display. Consequently, the users never need to leave their familiar hardware and operating system environments and can use distributed displays without external help.

Figure 1 – A distributed weather analysis and emergency warning system, a visualization scenario running on top of the ZAPP framework.

1. INTRODUCTION

Driven by the rapid evolution of computer simulations, acquisition technologies and computing hardware, datasets are growing at a rapid pace and even with advanced analysis and visualization techniques it becomes ever harder for a single user or specialist to interpret data alone. A method to allow for the real-time collaboration of multiple users is a distributed display system. A tiled display wall for example (Figure 1 right) provides significantly more area and resolution than a simple computer screen or projection and thus promotes the collaboration of multiple individuals. The ability to connect further display resources from remote locations, such as other large displays, simple workstations, or even mobile devices, provides a flexible platform for visual analysis. While the software systems required for such a scenario are very complicated, the user interfaces should not be. In this work, we do not focus on the user interface of a specific visualization application running on such a display, but on user interfaces to manage the display and its applications in general. To our best knowledge this part of the user experience has been largely neglected. While a number of very mature frameworks exist for communication, rendering, and synchronization in distributed display environments, administrators and users in addition require a control layer for installation, maintenance, and configuration and to select, start, and terminate applications. Our proposed framework was developed with the concept in mind that any user can launch and interact with an application on the display with minimal training and with no help by experienced staff controlling the display in the background, which we call “director of the institute proof”. In addition to this stable platform for everyday use of the display, we also want the system to support development of new applications without compromising the stability of existing software. With these goals in mind we developed the ZAPP framework.

The remainder of this paper is structured as follows: In the next section we will take a look at previous work for distributed displays. In Section 3 we give an overview on ZAPP. In the next section we outline ZAPP from a

usage perspective and thereby focus on three main areas – administration, development and application usage. Section 5 will then detail the implementation and architecture of ZAPP, focusing on the different components involved and the communication flow between them. We close the paper with a summary of the results and give directions for future work.

2. RELATED WORK

A substantial amount of research has been done in the area of distributed rendering, where partial results from each render workstation are either reassembled on a single display or routed to multiple displays. Humphreys et al. (2000, 2001) proposed WireGL, a framework for distributed OpenGL rendering on a cluster of workstations. On top of WireGL, Chromium (Humphreys et al. 2002) extends the framework with a more general approach to arrange the cluster workstations by utilizing a modular streaming model. Employing this model also removes the bottleneck of constantly transferring geometry required by render servers over the network, which made efficient fine-grained load balancing difficult with WireGL. Independently of the WireGL branch of systems, the Cross Platform Cluster Graphics Library (CGLX) has been developed (Doerr et al, 2011) specifically targeting distributed, high-performance visualization via a transparent OpenGL interface.
Equalizer (Eilemann et al., 2008) is another parallel rendering framework based on OpenGL with a focus on scalability, flexible configuration, i.e. supporting an arbitrary amount of workstations and displays, and a developer friendly programming model ("minimally invasive"). Parts of the framework are built on the OpenGL Multipipe SDK (Bhaniramka et al., 2005).

Allowing for both rasterization based approaches as well as ray tracing, DRONE (Repplinger et al., 2009) is a flexible framework for interactive visual applications rendering and displaying on multiple workstations.

SAGE implements a more general approach than the above frameworks, which focuses mostly on the rendering side (Renambot et al., 2006, Jeong et al., 2006, Nam et al. 2010). It is a distributed and scalable graphics framework with a focus on collaborative visualization allowing the exploration of large-scale scientific data sets, which may be rendered on a variable amount of workstations and displayed on a variable amount of displays, preferably on a tiled display wall. The graphics streaming employed in SAGE is built on previous work, TeraVision (Singh et al., 2004), which is a solution for high-resolution video streaming between arbitrary numbers of workstations. SAGE requires a high-bandwidth network to operate, limiting it to local collaboration. However it has been extended to support collaboration between multiple distant endpoints with Visualcasting (Jeong, 2009).

VTK is a visualization framework packaging a suite of visualization tools under a common interface. The framework has been extended to support cluster-based parallel rendering and delivering results to a single display or a display wall (Moreland and Thompson, 2003). The solution to render to a display wall is built on Chromium and IceT – implementations for both APIs are included. IceT is a parallel rendering framework targeting display walls as output. It is based on algorithms outlined by Moreland et al. (2001). Recently, Fogal et al. (2010) also used IceT to connect distributed memory multi-GPU clusters to visualize large datasets. More closely related to this work is a visualization framework described by Kim (2006) dubbed TileViewer, which supports multiple application areas including image viewing and video display. It includes a GUI to manage the displays and deploy files to the workstations.

In contrast to the above generic frameworks, which are designed to run various types of visual applications, domain-specific solutions exist which have been tailored to a particular area. Application areas include rendering and exploration of large geometric or volumetric data sets, high-resolution image display, video display and information visualization.

Vol-a-Tile (Schwarz et al., 2004) is a distributed volume rendering application able to display high-resolution data sets on a tiled display wall. Correa et al. (2002) present an extension to the iWalk out-of-core rendering system to enable distributed rendering of large static geometric data sets. Nam et al. (2009) describe the integration of ParaView into the SAGE framework to allow high-resolution rendering results to be visualized on a tiled display wall. JuxtaView (Krishnaprasad et al., 2004) is a distributed, parallel image viewer for ultra-high resolution image data. By distributing data across a cluster of render workstations and employing a caching and pre-fetching strategy to reduce the impact of network latency, arbitrary sized images can potentially be viewed interactively on tiled display walls. With Giga-stack, Ponto et al. (2010) propose a technique to explore multi-dimensional, giga-pixel images in a high-resolution display environment, such as a tiled display wall.

While the above generic frameworks and domain-specific solutions focus on their particular distributed rendering techniques, and thereby in some cases have proven to be an especially valuable contribution to this area, none focus on how to setup, manage and operate a large distributed display system from a usability perspective in a real-world, productive and possibly public environment. In this situation it is highly desirable for the system to be installed and accessed by non-expert users, i.e. for presentation purposes. Thus, we present ZAPP, a management framework to deploy, maintain and run distributed visualization applications in a flexible, stable and user-friendly way.

3. OVERVIEW

ZAPP is a lightweight management framework to run and collaboratively interact with distributed visual applications, which output to multiple displays, i.e. via a tiled display wall. The framework has been specifically designed to make development, deployment and maintenance of applications as well as setting up and configuring the framework and finally running applications easy, flexible and stable.

3.1 Basic architecture

Before starting with ZAPP, an initial hardware setup has to be available to deploy the framework on (see an example in Figure 2). This setup consists of a number of render workstations or nodes, each powering a single or multiple displays and running the distributed application. In addition, one machine must run control software; we call this machine the "control node". This control node is responsible for launching applications and for configuring the overall display alignment. It directly communicates with and keeps track of the available render nodes and their displays. Therefore, the render nodes register with the control node and periodically notify it about their current state. Render and control nodes are normally connected via a reliable high-bandwidth network. Furthermore, one or multiple machines can be used as UI nodes, which issue commands to the control node. UI nodes run a user interface to launch and terminate applications and, optionally, an application-specific user interface to interact with a running application. UI nodes may also be mobile devices connected wirelessly to the control node. Note that the control node may be a render and UI node at the same time, thus a minimal ZAPP configuration can run on a single machine. The different nodes and the software components running on them will be detailed in the following sections.

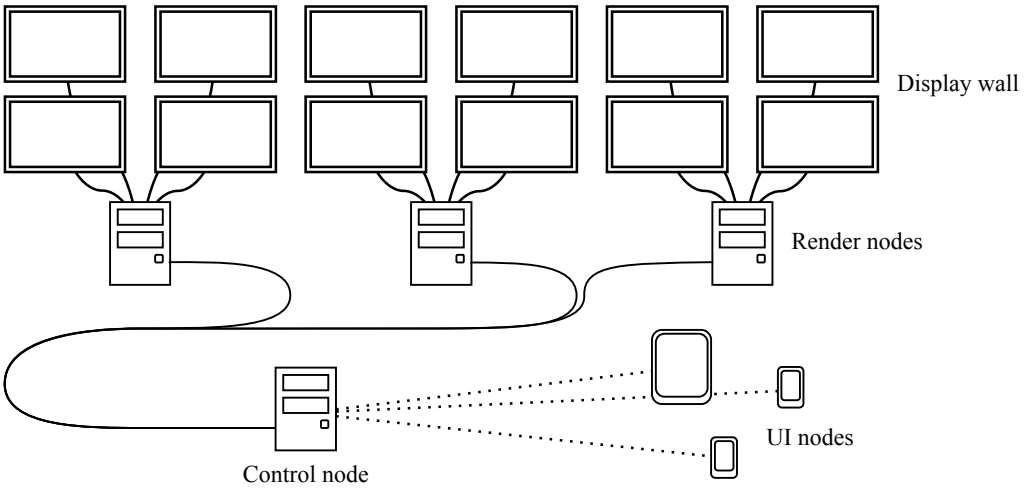

Figure 2 – Conceptual setup of a distributed display environment. Note that nodes do not necessarily correspond to separate systems, i.e. the control node can also be a render node.

4. USAGE

We will now detail the usage of ZAPP from the perspective of an administrator, a developer and a user who runs and interacts with applications.

4.1 Administration

Given the above hardware setup, ZAPP can be installed in a few steps. ZAPP provides an installer, which should be copied to a USB-device or a network share. First, this installer is used to prepare the control node. The initial setup is done using a GUI manager, which guides the installer through the individual steps. This involves setting up the ports the control node uses, defining an optional password, which is required to start and interact with applications, and restricting permission to a range of IP addresses. Note that all these steps may be skipped. In this case, default settings are chosen.

After installation, the control node process is a started and waits for render nodes to connect. The next step is to setup the render nodes, which can be done using the same installer that is used for the control node. This procedure can be fully automatic if the installer was run from a writable medium. In this case the control node's configuration was integrated into the render node installer. If this is not possible, another GUI lets the administrator configure these settings manually. Also, a name can be chosen for each render node, which will be

used to identify it at the control node. After the ZAPP process is running on a render node, it will automatically detect the number of displays attached to it and notify the control node.

The final step is then to configure the overall display alignment, also referred to as the display grid. This step is performed on the control node. There, each render node display can be mapped to a two-dimensional coordinate to define its position within the grid. A tile is defined as part of the grid identified by its coordinate and may be empty if there is no display linked to it. This can be used to account for holes in the physical display wall. In addition, the physical size of each display and its bezel can be set up individually or collectively for all tiles. Depending on an application's requirement, the bezel will either be considered a hidden part of the available display space or be ignored. Consequently, different kinds of tiles can be combined flexibly to setup a display grid using our framework. Of course, the recommended setup would be a grid representing a homogeneous tiled display wall.

After the alignment procedure is done, ZAPP installation is complete. Maintenance can be done at runtime. The render nodes will periodically confirm their presence and display setup with the control node. Should a render node or displays attached to it become unavailable, ZAPP will automatically detect this and disable the corresponding tiles in the display grid so application behavior can remain consistent. Adding render nodes can be done at runtime by repeating the render node installation process. At any time, the display grid can be edited to adapt to missing or additional displays using the manager GUI.

Installing ZAPP enabled visualization applications is straightforward. Application binaries and settings are packaged and can then be selected from the manager for deployment on the render nodes. An application can be flagged as experimental indicating it should not be used in a production environment yet. Afterwards, the application can be launched directly from the control node or indirectly through a UI node. The software to launch applications from a UI node has to be installed separately. Since a UI node never directly communicates with the render nodes, the only setup required is the address and port of the control node and possibly a password. The control node ultimately processes the user input and forwards it to the render nodes. Currently, ZAPP provides mobile launchers for Apple’s iOS device family, and an Android version is currently being developed.

4.2 Development

Every ZAPP application consists of two parts. The first is the server, which runs distributed on the render nodes and displays the visual content. The second is a client, which runs on the control node and is responsible for managing and synchronizing the application state, as well as issuing interactive commands from a user to the servers. A ZAPP application may have a third part, the UI, which runs on UI nodes and provides the user with an interface for interaction.

ZAPP provides several APIs to support developing each of these parts of an application. Templates integrating the APIs are available for the server, client and UI and can be used as a ready-to-go entry point for development. The network and synchronization API is used for communication between servers and clients as well as between clients and the UI. It is a platform-independent library for networking, which can be used in any context. The API includes functionality to synchronize the frame rate across the render nodes.

The display grid API is used on the client to query information about the render nodes and their display alignment. It maps a 2D grid coordinate to the corresponding render node and display so commands can be addressed to a specific tile transparently. Also, this API supports generating transformation matrices for each tile to represent a global transformation on the grid (i.e. to rotate a large image which spans several tiles). Considering a tile’s bezel can be optionally enabled when generating a matrix. Information about the current display grid will be automatically passed to an application on start-up. This API is also platform-independent.

The multi-monitor rendering API is available for the servers. It provides information for each available graphics adapter and the displays attached to it. An environment to render to multiple displays can be set up using this API, which has already been done for the server application template. Furthermore, the library provides an abstraction layer encapsulating common functionality of DirectX 10 and 11.

Finally, the mobile rendering API is used on the UI nodes. It encapsulates common functionality of OpenGL ES1.1 and 2 to support developing render features on the UI side of an application: an application may want to provide the user with a preview while rendering the final result on the display wall.

Note that using the aforementioned APIs and the application templates greatly simplifies creating or integrating existing applications, but none of these APIs are mandatory. Our framework is ultimately a management solution. Thus in the end, the developer is responsible for efficiently distributing an application across the available render nodes and displays, which may include distributing data, assigning render tasks, performing load balancing and choosing the render technique itself (i.e. rasterization or ray-tracing). In contrast, other generic frameworks (i.e. Humphreys et al., 2002, Eilemann et al., 2008) focus on the rendering side and features like data distribution or the render technique may be built-in.

Since ZAPP is able to fully operate on a single machine with any number of displays, development and debugging can be done locally before deploying the application to the production environment. An example application we developed is the TeraPixel Viewer, which allows exploring stacks of arbitrary sized images.

4.3 Using applications

While launching applications can be done directly from the control node, this is only intended for developers. A casual user's entry point is a UI node, which provides a launcher to start and terminate applications. A user runs the launcher, which has been set up to connect to the control node, and then selects and runs any of the available applications (see Figure 3). If an application provides a specific UI for interaction, this part is automatically started on the UI node. The UI of a ZAPP application has to be installed separately on every UI node to have access to the intended interactive features. Both launcher and UI are ideally deployed on a device familiar to the user. Note that the UI can also be started manually without the launcher to connect to an already running application. ZAPP does not limit the amount of UI nodes connecting. Consequently, an application may allow multiple users collaboratively interacting at once.

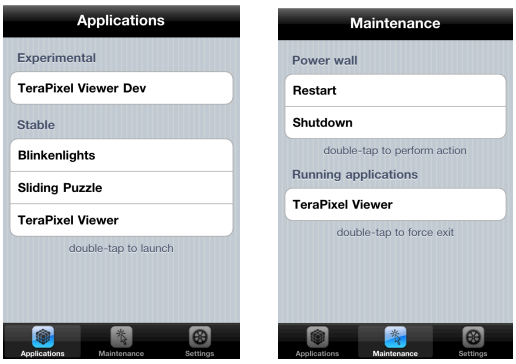

Figure 3 – Launcher UI running on an Apple iPhone. In these images the display of experimental apps was enabled and the TeraPixel application is currently running.

An additional feature of the launcher is to send a shut down command to the entire display hardware. This can be used to quickly power down the display wall after a presentation. It is recommended combining this feature with an automatic startup of ZAPP on the control and render nodes. This enables a user to boot the entire system by simply turning on the power.

To evaluate that the setup is indeed "director proof", we tested our mobile launcher installed on an iPod Touch with 15 non-expert users, who were asked to run the TeraPixel Viewer on a wall with 5x4 displays. No user had prior experience with the wall or ZAPP. Each user was able to browse through the list of applications and then select, run and interact with the TeraPixel Viewer. Especially users with prior iPod knowledge were able to finish the task instantly, while others were able to operate ZAPP with minimal instructions (i.e. explaining the touch display of an iPod).

5. IMPLEMENTATION

We will now detail the software components running on the different nodes and their communication flow with each other.

5.1 Control node processes

The control node is the center point of the ZAPP framework. It is responsible for managing the available render nodes and their displays as well as all configuration settings. It also runs the client part of an application, which maintains an application's state. The control node runs a persistent controller process, which is a TCP server listening for incoming connections from render nodes and UI nodes. Render nodes will establish a persistent connection to the controller and periodically confirm their presence and the displays they have available. Accordingly, the controller will update the display grid configuration. When a render node first connects, the controller will uniquely register it using the nodes name and physical address.

On initial start-up, the display grid is empty. An administrator is thus required to start another process, the manager GUI, to adapt the display grid to newly registered render nodes. Once a display grid is set up, the controller will automatically account for missing displays by deactivating the corresponding tiles in the grid. This happens if a render node or some of its displays become unavailable. Since there is no reliable way to identify a single display on a render node, should a display become unavailable, the controller will not know the exact tile linked to it and therefore deactivate the tile with the highest coordinate assigned to a display of the render node. Ultimately, the manager should be used to revise the display grid should a permanent change occur, i.e. a full column of displays of a display wall is removed or added, which would require resizing the grid. When

rebooting the system, the controller will automatically restore the tiles for a registered render node once it connects, so there is no need to setup the grid again.

The controller manages all settings in plain, human-readable text files. An expert user could thus quickly change settings without having to consult the manager. As the control node is the only point where the settings are held, a ZAPP configuration can easily be backed up and re-established by copying a few text files and replacing the files of another installation.

5.2 Render node processes

A render node is responsible for outputting visual content to its displays and thus runs the server part of an application. A render node runs a persistent daemon process in the background, which periodically iterates through the available displays and sends that information to the controller. The daemon is also a TCP server, which listens for connections from the controller. The controller will establish a persistent connection to this server once a render node has connected and issue commands to the daemon through it. These commands include launching an application, forcing an application to exit and shutting down the render node all together.

The daemon ZAPP currently provides is a Windows service building on DirectX to obtain information about the available displays. Future extensions to ZAPP will include a platform-independent solution.

5.3 UI node processes

A UI node is responsible to provide the user with an interface for interaction and thus runs the UI part of an application. There is no persistent process running here. The user can start a launcher process, which connects to the controller to request a list of available applications and the dimensions of the display grid. The request may or may not include applications flagged as experimental. An application can then be selected for launching. Note that by default an application is launched on the whole grid. However, it is perfectly valid to restrict an application to a sub-grid. Selecting a sub-grid for launching is not yet exposed in the mobile launcher GUI ZAPP provides for iOS devices.

Note that browsing through and launching applications can also be done directly from the manager GUI on the control node, but this is intended for developers only who are authorized to access the control node directly. In the following, we will focus on the scenario involving a UI node.

5.4 Communication flow when launching and running an application

Launching an application involves several steps (Figure 4). First, the launcher on a UI node sends a request to start a particular application on a sub-grid or the whole grid to the controller. In response, the controller starts the client process on the control node. It thereby passes information about the sub-grid, which for each tile includes the render node, and displays it is linked to, as well as the tile's bezel. A tile may be flagged as deactivated if it is not currently connected to a display. Some displays might be unavailable or there might be a physical hole in the display wall the client needs to know about to guarantee a consistent application behavior. Note that ZAPP can easily be extended to support launching clients remotely on a different machine than the control node. At the same time, the controller determines the render nodes whose displays are part of the sub-grid and then remotely starts the server process on each of these nodes. This is done by sending a launch command to the daemon running on each render node. This command includes information on the displays to use, which may be a subset of the available displays of a render node in case the application runs on a sub-grid, or some displays have not yet been linked to a tile. A daemon then starts the server process of the application.

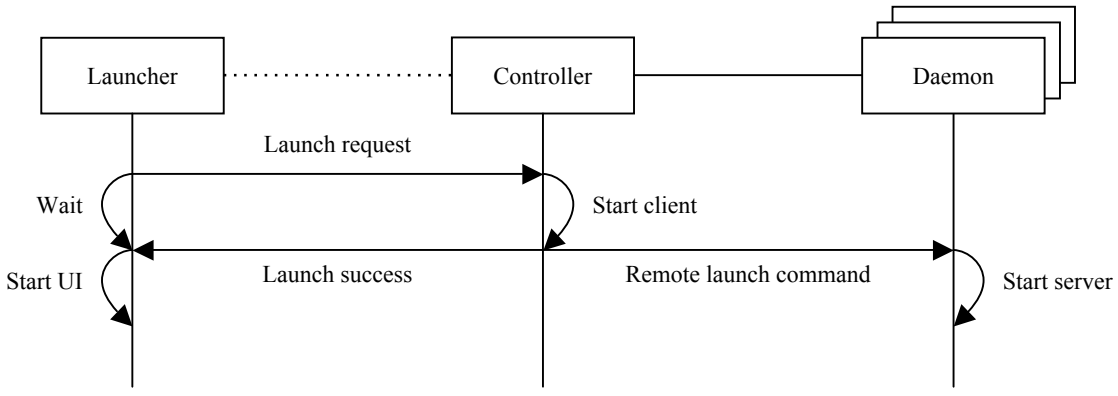

Figure 4 – Communication flow between controller, daemons and launcher when starting an application

Furthermore, the controller responds to the launcher to confirm the launch of client and servers. The launcher then starts the UI process so the user can start interacting with the application.

At this point, the involvement of controller, daemons and launcher is over and the application takes over. The servers and the UI establish a persistent connection to the client, which is the central point of communication and runs a TCP server of its own. From here, communication is application-specific and may be implemented in different ways. We will focus on the usual scenario (Figure 5). User input is sent from the UI to the client, which processes it, updates the application state accordingly and instructs the servers to update their displays. The client is especially responsible to keep the state of the distributed application synchronized. It therefore locks the application state until all servers have finished their work before accepting new input for processing.

In contrast to the reliable, high-bandwidth connection between client and servers, the connection between client and UI may be wireless and narrow-banded. Should a user get disconnected or close the UI while an application is running, a reconnection can be done at any time to regain control over interactive features. The UI will automatically attempt this if it loses connection.

To connect to the client, the clients address and listening ports were passed to servers and UI when starting them. Thus the controller is aware of the ports used by each application's client and attaches this information when remotely launching servers and UI. Should an application allow this, other users may now start the UI on their UI node to join and interact with the running application. Either the UI would directly allow the user to enter address and port of the client or the launcher can be consulted. The controller will confirm the running application with the launcher, which then starts the UI, passing the client information. For the UI node, it makes no difference whether the application was freshly started or already present. However, if a consecutive user has requested the application to launch on a sub-grid different to but overlapping the sub-grid the already running instance is using, the controller will reject the request. This brings us to the possible usage of multiple applications running at once. As long as their sub-grids do not overlap, it is perfectly valid to launch several applications or instances of a single one on the display grid. The space on the grid still available can be requested from the controller by the launcher. The controller will keep track of the applications running on which sub-grid and reject or accept new launch request accordingly.

An application is responsible for terminating itself gracefully and the UI should thus provide the user with a way to do this. An exit request issued through the UI is sent to the client, which forwards it to the servers. Since the controller needs to know about this to maintain the list of active applications, it holds a persistent, idle TCP connection to each client (a loopback connection, given that the client runs on the control node). A disconnection indicates an application has exited. An application can also be forcefully terminated through the launcher or the manager on the control node. This could be used to recover from a hanging application or should the UI get permanently disconnected from the client.

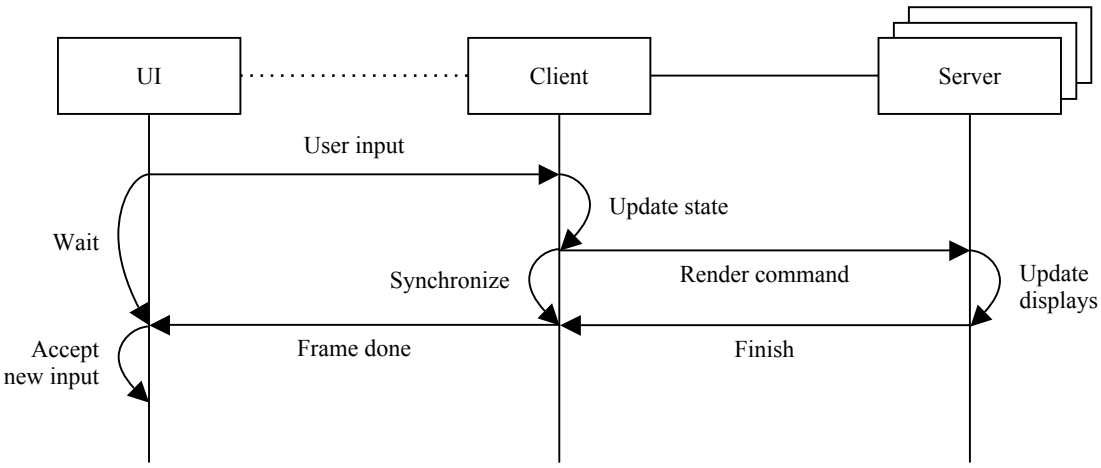

Figure 5 – Communication flow between client, servers and UI when an application has been started

6. CONCLUSION AND FUTURE WORK

In this paper we presented ZAPP: a management framework for distributed, high-resolution visualization systems. To our best knowledge this is the first solution that specifically targets the administration and launch process of programs on such a system. In a small informal survey we validated that arbitrary casual users are able to operate our framework with minimal instructions.

In the future we will continue to actively improve ZAPP. We will add an automatic system to layout display grids. This system will incorporate the camera found in mobile devices to automatically detect the alignment of displays. We are also preparing to distribute ZAPP as an open source application. Therefore, we will make sure that the entire framework can be run platform independently, i.e. we will provide an OpenGL based multi-monitor rendering API and extend the background process on the render node, which currently is a Windows

service, to be able to run as a POSIX daemon. In addition, we will provide our demonstrative applications such as the TeraPixel Viewer as open source.

ACKNOWLEDGEMENTS

This research was made possible in part by the Intel Visual Computing Institute, by the Cluster of Excellence "Multimodal Computing and Interaction" at the Saarland University, by the NIH/NCRR Center for Integrative Biomedical Computing, P41-RR12553-10 and by Award Number R01EB007688 from the National Institute Of Biomedical Imaging And Bioengineering. The content is under sole responsibility of the authors.

REFERENCES

Bhaniramka, P. et al., 2005, OpenGL multipipe SDK: a toolkit for scalable parallel rendering. *In Proceedings IEEE Visualization*. Minneapolis, MN, USA.

Correa, W. T. et al., 2002, Out-of-core sort-first parallel rendering for cluster-based tiled displays. *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*. Aire-la-Ville, Switzerland.

Doerr, K. and Kuester, F., 2011, CGLX: A Scalable, High-Performance Visualization Framework for Networked Display Environments. *IEEE Transactions on Visualization and Computer Graphics*. Volume 17 Issue 3.

Eilemann, S. et al., 2008, Equalizer: a scalable parallel rendering framework. *ACM SIGGRAPH ASIA 2008 courses*. Singapore, Singapore.

Fogal, T. et al. 2010, Large Data Visualization on Distributed Memory Multi-GPU Clusters, *High Performance Graphics*, Saarbrücken, Germany

Humphreys, G. et al., 2000, Distributed rendering for scalable displays. *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*. Washington, USA.

Humphreys, G. et al., 2001, WireGL: a scalable graphics system for clusters. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York, USA.

Humphreys, G. et al., 2002, Chromium: a stream-processing framework for interactive rendering on clusters. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. New York, USA.

Jeong, B. et al., 2006, High-performance dynamic graphics streaming for scalable adaptive graphics environment. *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*. New York, USA.

Jeong, B., 2009, Visualcasting – scalable real-time image distribution in ultra-high resolution display environments. Dissertation, University of Illinois, Chicago, IL, USA.

Kim, S-J., 2006, *The diva architecture and a global timestamp-based approach for high-performance visualization on large display walls and realization of high quality-of-service collaboration environments*. California State University, Long Beach, CA, USA.

Krishnaprasad, N. K. et al., 2004 JuxtaView - a tool for interactive visualization of large imagery on scalable tiled displays. *Proceedings of the 2004 IEEE International Conference on Cluster Computing*. Washington, USA.

Moreland, K. et al., 2001, Sort-last parallel rendering for viewing extremely large data sets on tile displays. *Proceedings of the IEEE 2001 symposium on parallel and large-data visualization and graphics*. Piscataway, USA.

Moreland, K. and Thompson, D., 2003, From Cluster To Wall with VTK. *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics*. Washington, USA.

Nam, S. et al., 2009, Remote visualization of large-scale data for ultra-high resolution display environments. *Proceedings of the 2009 Workshop on Ultrascale Visualization*. New York, USA.

Nam, S. et al., 2010, Multi-application inter-tile synchronization on ultra-high-resolution display walls. *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*. New York, USA.

Ponto, K. et al., 2010, Giga-stack: A method for visualizing giga-pixel layered imagery on massively tiled displays. *Future Generation Computer Systems*. Volume 26 Issue 5.

Renambot, L. et al., 2006, Collaborative visualization using high-resolution tiled displays. *ACM CHI Workshop on Information Visualization Interaction Techniques for Collaboration Across Multiple Displays*. Montreal, Canada.

Repplinger, M. et al., 2009, DRONE: A Flexible Framework for Distributed Rendering and Display. *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part I*. Berlin, Germany.

Schwarz, N. et al., 2004, Vol-a-Tile - A Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays. *Proceedings of the conference on Visualization '04*. Washington, USA.

Singh, R. et al., 2004, TeraVision: a distributed, scalable, high-resolution graphics streaming system. *Proceedings of the 2004 IEEE International Conference on Cluster Computing*. Washington, USA.