Powerwalls and Scalability: Implementation Issues

Ulrich von Zadow
Archimedes Exhibitions GmbH
10405 Berlin, Germany
uz@archimedes-exhibitions.de

Ricardo Langner
Interactive Media Lab
Technische Universität Dresden
01062 Dresden, Germany
langner@acm.org

Ulrike Kister
Interactive Media Lab
Technische Universität Dresden
01062 Dresden, Germany
ukister@acm.org

Raimund Dachselt
Interactive Media Lab
Technische Universität Dresden
01062 Dresden, Germany
dachselt@acm.org

Copyright is held by the author/owner(s).
CHI 2013 Extended Abstracts, April 27May 2, 2013, Paris, France.
ACM 978-1-4503-1952-2/13/04.

Abstract
The very high resolutions supported by powerwalls and other large display configurations pose an implementation challenge, since rendering of the display contents must be distributed to achieve sufficient performance. In this position paper, we describe several strategies for implementing corresponding middleware and overcoming this challenge. We examine the strategies considering the two implementation goals network transparency and scalability. Using goals and strategies as context, we analyze prior work in the area and present a number of experiences in implementing commercial interactive surfaces that use distributed rendering.

Author Keywords
Powerwall, Scalability, Multiple Displays, Display Walls, Middleware, Rendering

ACM Classification Keywords
H.5.m [Information interfaces and presentation (e.g., HCI)]: Miscellaneous.

Introduction
Large displays have been used successfully in many different application areas, e.g., command and control, vehicle design, geospatial imaging [2, 6] and scientific visualization [5]. In these application areas, it is very

common for users to work with large amounts of data. Correspondingly, they benefit immensely from the very high resolutions possible using wall-sized displays. With these resolutions, however, comes an implementation challenge: The contents of the display - potentially 100 million pixels or more - need to be computed with adequate performance. At the same time, development time is a factor, so application-level code should not have to be concerned with achieving this level of performance.

We will examine requirements and implementation strategies for powerwall middleware. Given the above, two important goals should be:

- Scalability: Performance should not depend on the number of displays attached and
- Network Transparency: Application-level code should not need to know that this is a distributed system.

In general, sufficient computing power to achieve scalability is present, since display walls usually run on a cluster of PCs and there is a small, fixed number of displays per computer. The issue, then, is in harnessing this computing power. Amdahl's Law [1] gives us a guideline here: We need to maximize parallelism and minimize those sections of the rendering pipeline that run serialized.

The remainder of the paper will examine a number of feasible strategies in this context. We start with an overview of prior work, then describe the aforementioned implementation strategies. Practical experiences with several display wall implementations are the focus of the next section. We conclude with a discussion and pose a number of open research questions.

Prior Work

Much work has been done in parallel rendering and supporting multi-display systems. Ni et al. [6] provide a survey, while Staadt et al. [7] do a performance analysis. Several scene graph libraries such as OpenSG [8] support parallel rendering of 3D scenes. In general, these libraries can provide high scalability in this application domain as long as efficient culling – discarding of geometry based on visibility – is possible. The popular Chromium [4] library allows parallel execution of arbritary OpenGL code. This makes it usable for a wide variety of applications. Unfortunately, the current version only supports OpenGL 1.5 (at the time of writing, the latest version is 4.3). In these systems, the application itself runs on a single computer and rendering is distributed to the cluster. The VR framework VRJuggler [3] takes a different approach: The application runs on all computers, execution is synchronized and user input distributed to all nodes.

Using prior work as well as our own experiences in implementing multi-display renderers as basis, we have identified a number of possible implementation strategies. These are described in the following section.

Implementation Strategies

This following section describes a number of possible approaches and delienates corresponding consequences with respect to the system goals of scalability and network transparency. It also places the prior work described above in this framework.

Variant 1) Central Master, Rendering on Master
The application runs on a central computer, the master. All user input is routed to this computer, which runs application-level code and does all rendering at full resolution. The resulting images are distributed over the

Figure 1: Famous Grouse Experience interactive floor.

network to display slaves that only place the respective images on the screen. A positive aspect of this method is that it is easy to achieve network transparency. On the flip side, it is obvious that the approach does not scale at all, since the central computer has a very high working load. In addition, the transfer of complete display images consumes very large amounts of network bandwidth even if compression is taken into account. Still, depending on the application area, this can be a feasible strategy because it is easy to realize. For instance, the 'wall' video filter of the VLC video player[1] is implemented in this way.

Variant 2) Synchronized Applications, Full Scenes
In this approach, there is no central master. An identical application runs on all computers, all input is broadcast to the complete cluster and the complete scene is processed on all of them as well. However, each computer only renders the subset of the scene that is actually visible on its display. Network transparency is achieved only in part, because application code needs to be completely deterministic for this to work – timing-dependent or random behaviour will cause loss of synchronization. This approach scales well to a certain extent, since rendering is done in parallel. It is also relatively straightforward to implement. VRJuggler [3] is realized in this way but uses an additional central master that is only responsible for synchronization.

Variant 3) Synchronized Applications, Partial Scenes
Similar to variant 2), there is no central master. An identical application runs on all computers, but in contrast to 2), each application instance only handles the part of the scene that it is displaying. Input is also handled locally, and synchronization of scene content is done at application level. Because of this, there is no network transparency. In principle, this approach leads to the same issues with nondeterministic applications as 2), but application-specific code can mitigate the effects. On the positive side, since even the application code runs in parallel, this variant scales better than all other solutions.

Variant 4) Central Master, Distributed Rendering
In this approach, there is one central master that runs the application and handles user input. Rendering is prepared on this computer as well. The scene is then distributed and display slaves are responsible for rendering their individual parts. This strategy provides full network transparency. It also scales fairly well, since the rendering is done in parallel.

OpenSG [8] and several other scene graph libraries, as well as the Chromium OpenGL wrapper [4], implement this method. The implementations differ in the exact way that work is distributed and the amount and type of work done on the master. For instance, Chromium transfers the complete scene geometry in every frame, limiting performance for complex scenes, but it only distributes geometry to those nodes that will actually display it. OpenSG does the opposite: Each node receives the complete scene, but only geometry changes are distributed [7].

Realization Examples

In the following section we will describe examples of large display implementations in which we took part, explaining challenges and issues that we faced. We place these examples in the framework delienated in the previous section and examine the solutions with respect to the goals presented in the introduction.

[1] http://www.videolan.org/vlc/

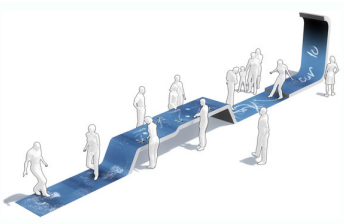

Figure 2: Layout of the O2 Sculpture

Famous Grouse Experience
The Famous Grouse Experience is an interactive environment at the Glenturret Distillery in Creif, Scotland, that we implemented in 2002 at ART+COM[2]. The installation features an interactive floor realized using six projectors - figure 1 shows it in use. A central master processes user input and sends input events as well as high-level application state messages to six display servers. The display servers are application-specific; additional peer-to-peer messages are sent to synchronize low-level application state. For instance, one application phase features a water simulation. This simulation runs in a distributed fashion and water level data is exchanged between adjacent display servers.

The setup combines elements of variants 3 and 4. Since all networking and synchronization is handled by the application, we were able to distribute the work very effectively. On the other hand, the logic is complex: master-slave as well as peer-to-peer networking is implemented at the application level and interspersed with other application code. The unclear separation of concerns led to code that was difficult to understand and a correspondingly high development time.

O2 Sculpture
The O2 Sculpture we implemented at ART+COM in 2004 is the central element in the flagship store of the telecommunications provider O2 in Munich[3]. It consists of an 18 meter long interactive strip that has floor, table and wall parts - shown in figure 2. Twelve ceiling-mounted projectors are used as displays. The installation features fairly involved interaction and a complex 3D scene.

[2]http://www.artcom.de/en/projects/project/detail/grouse-experience/
[3]http://www.artcom.de/projekte/projekt/detail/o2-skulptur/

Among the scene elements are ice that cracks when stepped upon and O2-typical water bubbles as well as rendered cell phones and videos (see figure 3). This installation was realized in a fashion similar to variant 3: There is no central master and input is processed locally. Each computer has the complete scene in local storage but only modifies and renders those elements which it displays. Scene changes are then broadcast and replicated across the cluster. Again, we were able to distribute work very effectively, but application-level synchronization led to a high development time.

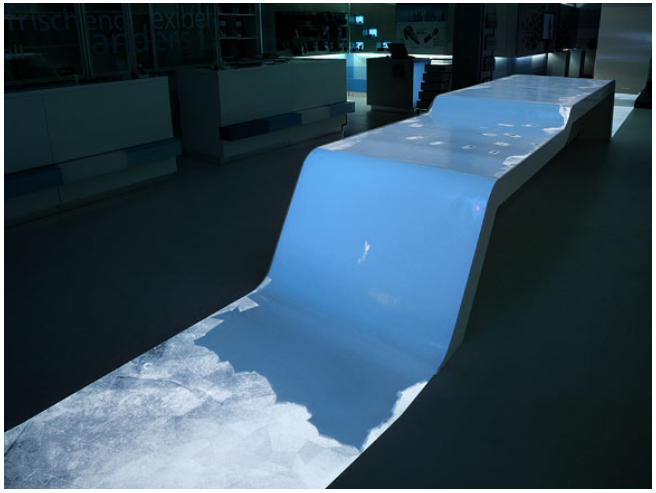

Figure 3: O2 Sculpture in idle state.

Max Planck Science Gallery Berlin
At Archimedes Exhibitions, we developed and deployed a six display multitouch wall at the Max Planck Science Gallery in Berlin in 2012[4] (see figure 4). Content elements

[4]http://www.archimedes-exhibitions.de/exhibitions/highlights/_/max-planck-science-gallery.html

Figure 4: Multitouch wall at the Max Planck Science Gallery

are images, videos and text snippets that form a narrative and move from one display to the next. The implementation was based on the media development framework libavg[5]. The parallelization strategy corresponds to variant 2, with the entire scene processed on each computer, user input broadcast across all displays and a cropped scene rendered locally.

Even though, e.g., video decoding is duplicated on all nodes, the installation has ample performance reserves. In large part, this is due to the fact that libavg performs video decoding in parallel on different cores of the computer and utilizes hardware decoding where possible as well. However, it does not scale. Network transparency is not fully achieved, because realizing a fully deterministic application is significant work: Subtle timing variations can lead to wildly different application states in extreme cases.

[5]https://www.libavg.de

Conclusion and Future Work

We have examined a number of implementation strategies for supporting very high-resolution displays with regard to the two goals scalability and network transparency. In addition, we have described several real-life implementations of distributed rendering systems and analyzed them in relation to the same goals.

We are interested in exploring this further. Variant 4 (distributed rendering with a central master) seems to be the only approach that couples network transparency with sufficient scalability, and there seem to be a number of open questions along this route. As an initial step, it would be interesting to have more information on the types of loads that typical powerwall applications generate: What processing power does the application logic need? Is there a high CPU load because of geometry processing? How many graphics primitives are there? Does the geometry change regularly? Is a large amout of fragment processing power needed? Is video display (or are other forms of display streaming) needed? In other words: Where are the bottlenecks? Different application

domains - command and control, vehicle design, geospacial imaging, scientific visualization, etc. - will all have different answers to these questions, and they can be used to determine requirements for middleware.

Given these requirements, it should be possible to evaluate existing solutions in-depth: How do they cope with typical loads across the rendering pipeline? How does this match to the requirements of the different application domains? Both questions can be explored on a conceptual level and by crafting appropriate benchmarks. Finally, these insights could be used to build new, more powerful middleware that combines full network transparency with a maximum amount of scalability.

Acknowledgements

We would like to thank our colleagues at Archimedes and ART+COM for the work on the distributed rendering systems described.

References

[1] Amdahl, G. M. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, AFIPS '67 (Spring), ACM (New York, NY, USA, 1967), 483–485.

[2] Ashdown, M., Tuddenham, P., and Robinson, P. High-resolution interactive displays. In *Tabletops*, C. Müller-Tomfelde, Ed., Human-Computer Interaction Series. Springer, 2010, 71–100.

[3] Bierbaum, A., Hartling, P., Morillo, P., and Cruz-Neira, C. Implementing immersive clustering with vr juggler. In *ICCSA (3)*, O. Gervasi, M. L. Gavrilova, V. Kumar, A. Laganà, H. P. Lee, Y. Mun, D. Taniar, and C. J. K. Tan, Eds., vol. 3482 of *Lecture Notes in Computer Science*, Springer (2005), 1119–1128.

[4] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D., and Klosowski, J. T. Chromium: a stream-processing framework for interactive rendering on clusters. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, ACM (New York, NY, USA, 2002), 693–702.

[5] Isenberg, P., Isenberg, T., Hesselmann, T., Lee, B., von Zadow, U., and Tang, A. Data visualization on interactive surfaces: A research agenda. *IEEE Computer Graphics and Applications 33*, 2 (2013). To appear.

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

[7] Staadt, O., Walker, J., Nuber, C., and Hamann, B. A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *Proceedings of the workshop on Virtual environments 2003*, ACM (2003), 261–270.

[8] Voß, G., Behr, J., Reiners, D., and Roth, M. A multi-thread safe foundation for scene graphs and its extension to clusters. In *Proceedings of the Fourth Eurographics Workshop on Parallel Graphics and Visualization*, EGPGV '02, Eurographics Association (Aire-la-Ville, Switzerland, Switzerland, 2002), 33–37.