Eurographics Symposium on Parallel Graphics and Visualization - Short Papers (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

A Simple, Distributed Rendering Toolkit for Multi-Screen, Rendering Clusters

A. Tetzlaff[1] C.-A. Bohn[1] T. F. Horn[2]

[1]Wedel University of Applied Sciences
[2]Planetarium Hamburg

Abstract

We present a cluster-based rendering system for driving arbitrary multi-screen environments. Essentials of the system are that there is no sophisticated synchronization hardware required; applications can be developed on desktops and then are able to be copied on nearly any of such environments; the number of rendering nodes is arbitrary scalable for driving any required resolution or any number of projection screens.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Distributed Rendering Systems

1. Introduction

In recent years, with the development of graphics hardware users' demand in highly advanced graphics output became a crucial issue. Since display sizes and resolutions seem to meet their physical limits, the natural way to overcome this limitation is to simply increase the number of displays and combine them physically in a way that they imitate one large screen.

Some early attempts are the well-known double-screen desktop environments. More recent technologies include *Virtual Reality* Technologies (i.e., *head-mounted displays* [FMHR87]), multi-screen training- (i.e. flight-, drive-) simulators, multi-projector screens like *CAVE*-like environments [CNSD93], high-resolution *Power Walls* [HH99], and dome-like cinemas like *IMAX* theaters and planetariums.

Vital challenge in developing applications for multi-screen technologies is to handle the underlying rendering hardware. Whereas two-screen environments often can be realized by a single computer feeding one single graphics card, more sophisticated installations often base on the use of several PCs. Developers have to make strong efforts for parallelizing the application and the rendering.

In this work we describe a system applicable to common multi-screen environments, which is tested in a concrete installation — the Planetarium Hamburg [pla]. This planetarium consists of seven rendering PCs driving seven projectors forming together one spherical projection dome.

Figure 1: *The Planetarium-Hamburg — test case for the presented approach.*

Usually, access to soft- and hardware of planetariums for experimental applications is very restricted. Often they only run commercial productions which are pre-rendered and just "played". Here, parallelization is realized by a kind of a "parallelized video player". Thus, incorporating own (*OpenGL*-) applications must happen by avoiding massive infringement on the daily operational demands of such an environment and on secrecy issues from its hard- and software supplier.

In this work, we show how such a rendering system can be realized without any change in the aimed target hardware, without touching the software configuration, and without a vast amount of work put into parallelization issues of render-

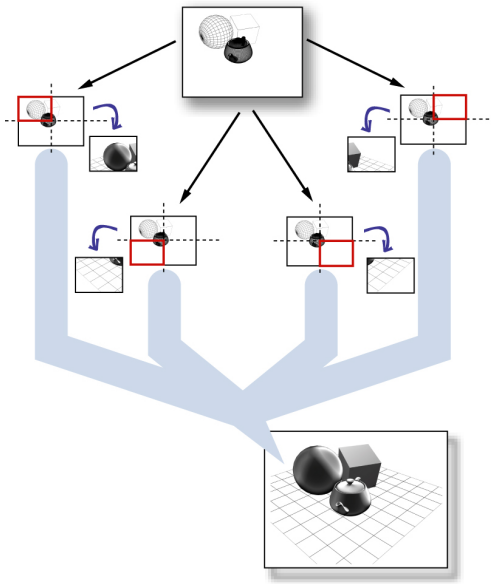

Figure 2: *Partitioning of one screen into several sub-screen connected to a single rendering node each. Nodes are separate programs, according to the master-slave clustering model, running on the same or different platforms.*

ing tasks. Any pre-compiled *OpenGL*-application (the binaries) just need to be copied and executed on each rendering node.

Several alternative software packages for distributed rendering are available, only mentioning some of them like *Chromium* [HHN*02], *OpenSG* [osg], the *CAVE library* [cli], *VR Juggler* and *Net Juggler* [jug], or *Avango* [avg]. A great comparison of these systems can be found in [vrc].

In the following we shortly outline the basics of our system, present its vital advantages compared to classical approaches and finally we will conclude with a summary and show two example applications.

2. Approach

Porting applications written for *one* graphical output to a multi-screen environment is not a trivial task. Considering the simplest case — one graphics card on one computer drives two screens — the programmer must handle two framebuffers. Going steps further would become harder — two graphics cards need a parallelization of the rendering tasks, two computers with own graphics hardware requires additional parallelization of separate processes, additional synchronization and network communication.

2.1. Structure of the Rendering Cluster

The underlying system consists of a set of rendering nodes which create the final picture by rendering each a certain cut-out of the whole spherical screen (see Figure 2). The nodes configuration can be classified according to [SWNH03] as a typical *master-slave* cluster. The complete scene data is available on every rendering node. The animation and rendering status of each of them is controlled by one master node which is also responsible for tracking and distributing user interaction. In this work the term “node” is not connected to a certain hardware platform, moreover rendering nodes and the master node are just separate programs which may run on different platforms but also as concurrent tasks on the same platform. “Distributing an application” simply means taking the *OpenGL* program and executing it *n*-times as *n* separate tasks on one or more separate rendering platforms.

After the start each rendering node first connects to the master node. The master node then triggers the rendering and “supervises” the rendering nodes in the following time. Input events arising at the master’s periphery for, i.e., modifying the viewing parameters, are passed to each rendering node. Only in case of user interaction like key strokes or mouse movements the according event data is transfered. The amount of this data is negligible, thus we realized the transfer by using common network transfer technology — a 100 MBit network, which even was the only available at our test environment at the planetarium.

2.2. Synchronization

The separate pipelines must display their image parts in a synchronized manner. In our application we are satisfied with a maximal delay of one frame on separate rendering nodes. Thus, the approach is capable of driving all non-stereo multi-screen systems, like in our application the Planetarium-Hamburg, all active stereo systems with a specialized hardware synchronization like [dig], or all passive stereo systems.

The basics of our synchronization are as follows. In advance to the start of the actual application all nodes of the cluster have to connect to the master via *TCP*. The master listens for the rendering nodes at a predefined port to receive their local configurations. Then it provides them with his demand in rendering and simulation speed in terms of a certain value of *frames per second (FPS)*. Up to now the transfer is not time-critical and thus delivered through *TCP*.

To start the animation the master sends a start-command via *UDP* to all nodes. Now, by choosing *UDP* we abandon the flow control overhead of *TCP* enabling short transport times for time-critical data like the start-command. These transport times virtually are negligible. Then rendering nodes start at a predefined status — synchronized except for the delay the UDP transfer causes. Through several tests

we proved that the delay is clearly below one frame and thus negligible.

After initial synchronized triggering the nodes compute the animation parameters and finally render their frustum of the scene. Herewith, every node tries to maintain the frame rate demanded by the master node, i.e. if the master wants a 40 frames per second rate, the rendering node waits until a full 1/40 of a second went by until the next frame is generated.

In other words, our synchronization is realized by the internal clocks of each rendering node. This is uncommon but satisfies our application conditions excellently. It leads to the fact that we virtually do not have synchronization transfer overhead.

Of course a lack of synchronization may arise due to the following reasons.

- The master's distribution of the current frame number at the start is delayed.
- The clock rate on separate nodes differ.
- Some nodes cannot achieve the requested rendering and animation speed — they "arrive too late" after one rendering task.

To avoid the first two cases "re-synchronization"-packets are sent every few seconds. In all cases the according nodes react in the same manner. They try to catch up to the correct frame number by skipping the rendering part and just proceeding with the integration part of the animation.

This minimal amount of transfer makes it possible to scale the number of rendering nodes arbitrary without reducing the rendering performance.

2.3. Handling User Events from the Master Node

Events recorded by the master node have to be delivered to the rendering nodes. Herewith, it must be guaranteed that all nodes react on events at the same time — the same frame number — otherwise the animation diverts on different nodes. To avoid that, the master node collects and serializes all events of the current frame, marks this event set by the current frame number an distributes it via *UDP* packets to the rendering nodes. Each of them stores the dataset temporarily and triggers the contained events not until it reaches the frame number which the master attached at the event set. Thus a consistent animation is guaranteed even if nodes lag a frame due to their local rendering resources. By incrementing the assigned frame number before sending an event set it is guaranteed that even nodes which are one frame ahead will register the events at the right frame. Although this results in a minimal latency of user interactions it is preferable compared to an inconsistent animation.

In our desktop environment and also in the planetarium's PC cluster, it has been shown that variations of up to two frames do not affect the visual quality.

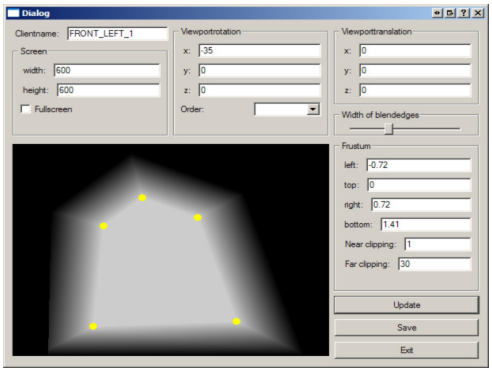

Figure 3: *The user interface to define viewing and blending parameters of overlapping projection areas illuminated from different rendering nodes.*

2.4. Edge Blending

Projections physically combined to large displays typically overlap at the borders to avoid cracks in the whole projected area. At these overlapping areas the generated image would appear brighter if it is not dimmed accordingly at each participating projector output. Since only few projectors have a *hardware edge blending* capability to realize this adaption, we added edge blending by software implemented through using the blending facilities available on common graphics cards.

2.5. Rendering Node Configuration

The basic configuration — parameters like shape, orientation, and position of the frustum — of each node of a multi-screen environment is specified by one *XML*-file per node. It is parsed by the rendering application at startup and sent to the master application when logging in. Here, also the edge blending cut-out is defined by an arbitrary polygon which can be defined through a graphical configuration interface (see Figure 3).

3. Results and Summary

Our proposal enables the user to write multi-screen environment applications fairly easy and straight-forward.

- *Implementation:* The user simply writes an *OpenGL* program on a usual computer with one graphics pipeline. Then, porting the application to the multi-screen environment means to copy it to all participating rendering nodes only — not even another linkage is necessary.
- *Synchronization:* Synchronizing the rendering nodes is accomplished every few seconds. Inbetween, synchronized rendering is guaranteed through the internal clocks of each node. With this attempt, only negligible data exchange between the master and the rendering nodes arises,

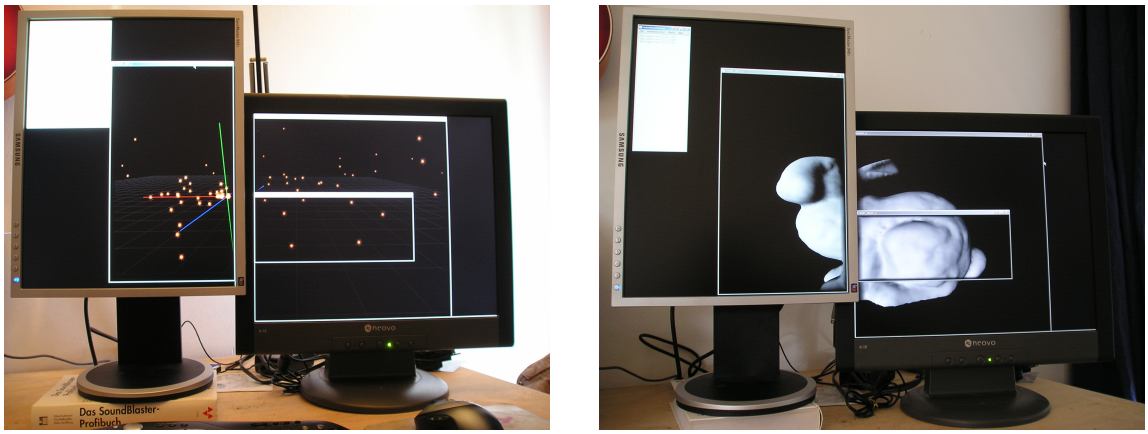

Figure 4: *Test applications (a particle system and a bunny model) of the presented system. Several separate rendering processes running on each of two PCs.*

which makes it possible to realize a rendering cluster of a theoretically unlimited size. Another advantage is that the system does not need sophisticated network hardware — not even a gigabit network — to guarantee synchronization between master and rendering nodes.

- *Test case:* We tested our software developed on a single screen desktop PC in a professional, commercial environment. It worked from scratch without any re-compilation proving its portability, simplicity, and robustness.

Figure 4 exposes two example renderings of a simple particle system and the well-known bunny model. The software of each of the shown implementations consists of just about 300 lines of source code. By using *Qt4* [tro] for high-level access to GUI, network, and threads, the framework is capable of being compiled under Windows and Linux. Adaption to any other hardware environment (like *Cones*, *CAVE*s, or *Domes*) only requires the modification of the frustum and the edge blending parameters on each rendering node. The development process of an application may be completely decoupled from the target-environment.

References

[avg] http://www.avango.org.

[cli] http://www.vrco.com.

[CNSD93] CRUZ-NEIRA C., SANDIN D. J., DEFANTI T. A.: Surround-screen projection-based virtual reality: the design and implementation of the cave. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), ACM Press, pp. 135–142.

[dig] http://www.digital-image.de.

[FMHR87] FISHER S. S., MCGREEVY M., HUMPHRIES J., ROBINETT W.: Virtual environment display system. In *SI3D '86: Proceedings of the 1986 workshop on Interactive 3D graphics* (New York, NY, USA, 1987), ACM Press, pp. 77–87.

[HH99] HUMPHREYS G., HANRAHAN P.: A distributed graphics system for large tiled displays. In *VIS '99: Proceedings of the conference on Visualization '99* (Los Alamitos, CA, USA, 1999), IEEE Computer Society Press, pp. 215–223.

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., AHERN S., KIRCHNER P. D., KLOSOWSKI J. T.: Chromium: a stream-processing framework for interactive rendering on clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), ACM Press, pp. 693–702.

[jug] http://www.vrjuggler.org.

[osg] http://www.opensg.net.

[pla] http://www.planetarium-hamburg.de.

[SWNH03] STAADT O. G., WALKER J., NUBER C., HAMANN B.: A survey and performance analysis of software platforms for interactive cluster-based multi-screen rendering. In *EGVE '03: Proceedings of the workshop on Virtual environments 2003* (New York, NY, USA, 2003), ACM Press, pp. 261–270.

[tro] http://www.trolltech.de.

[vrc] http://chromium.sourceforge.net/doc/llnlcopy.html.