IPT & EGVE Workshop (2005)
R. Blach, E. Kjems (Editors)

A Game Engine-based Multi-Projection Virtual Environment with System-Level Synchronization

Naoki Hashimoto, Yoshihiko Ishida and Makoto Sato

Precision and Intelligence Laboratory, Tokyo Institute of Technology, Japan
E-mail: naoki@hi.pi.titech.ac.jp

Abstract

In multi-projector displays, which surround users with high-resolution images, a PC-Cluster is often used for realistic and real-time image generation. However, developing applications that support parallel processing on the PC-Cluster is quite troublesome. It is also difficult to acquire sufficient rendering performance because of the limited bandwidth of the PC-Cluster. Therefore, we aim to achieve affordable and accessible software environments for the multi-projector displays. In this paper, we describe a self-distributing software environment for inheriting existent game engines which provide basic functions of realizing virtual environments. This environment achieves minimum data communication based on a master-slave model. The communication mechanism is automatically applied to target applications by intercepting APIs. Hence we can directly exploit high-capability of the existing game engines on the multi-projector displays.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems C.2.4 [Computer-Communication Networks]: Distributed Systems I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realismn

1. Introduction

Recently, a number of immersive projection displays have been developed that surround users with high-resolution and stereoscopic images [CNSD93]. Such displays are used as display devices for Virtual Reality (VR). Immersive projection displays require multiple projectors, for high-resolution image projection, and a PC-Cluster, which is a set of PCs connected by a commodity network, for real-time 3-D Computer Graphics (CG) generation. Today's rapid evolution of PC and projector technologies has greatly contributed to achieve powerful and high-quality display systems.

On the contrary, recent VR applications have been making a challenge to use game technology for affordable and accessible virtual environments [JL05]. Computer games with the most advanced simulation and graphics usually adopt a *game engine*, a commercially available software package that provides basic functions for realistic 3-D graphics, a built-in physics engine and robust networking for shared environments. This technology is quite helpful to achieve high-quality virtual environments.

However, most of VR software resources including a *game engine* have been developed for a stand-alone PC environment. Although the latest hardware facilitates the achievement of the multi-projector displays, the software that controls the hardware and creates 3-D contents requires additional support in order to operate such special hardware using multiple PCs and projectors. In the display system using a PC-Cluster, distributed-memory type parallel processing between PC nodes is essential, for basic surface-based CG generation. When these software resources are used on PC-Cluster-based display systems, their source codes must be modified and special software that is adapted to PC-Cluster-based systems must be developed. In addition to the extra development costs, we also encounter difficulty in acquiring the source codes of existent applications, including useful and commercial software. The network bandwidth of a common PC-Cluster also causes performance problems. In parallel processing, the network bandwidth strictly governs the overall performance. Common PC-Clusters have 100 Mbps Ethernet, which is relatively narrow for parallel processing. Therefore, in PC-Cluster-based display systems,

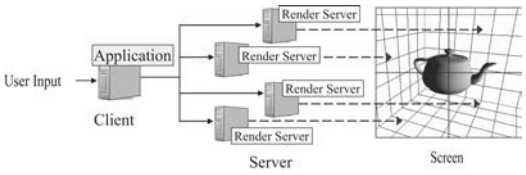

Figure 1: *Client-Server model.*

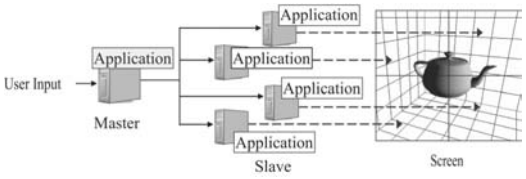

Figure 2: *Master-Slave model.*

effective communication processes between PC nodes are required for real-time image generation.

In this paper, we aim to achieve *game engine*-based multi-projection virtual environments. In order to utilize existing *game engines*, we introduce a self-distributing software environment for system-level synchronization without any additional modifications. This environment achieves a low-latency communication model with 100 Mbps Ethernet, and automatically distributes existent applications over all of the PC nodes. Using this software environment, we can easily inherit *game engines* technology, and extend the applicability of multi-projector displays.

2. Previous Works

Several software development and execution environments have been proposed for multi-projector displays using a PC-Cluster. As parallel processing architecture, they are categorized into two models, a Client-Server model and a Master-Slave model [SWNH03].

In the Client-Server model shown in Figure 1, applications are executed on a client node and rendering information, such as *OpenGL* or *Direct3D* command stream, is generated. The information is sent to server nodes and is used for image generation. *WireGL* [HEB*01] and *Chromium* [HHN*02] are famous implementations using this model. They achieve parallel rendering by replacing an *OpenGL* driver with a special driver that can automatically distribute rendering commands to the server nodes. This special driver has the same APIs as the standard *OpenGL* driver. Hence, *OpenGL* applications can be used without any special modifications of their source codes. However, in large, complicated 3-D scenes used in practical applications, the amount of communication data between the client and the server increases dramatically as the network bandwidth of the PC-Cluster becomes a bottleneck of total rendering performance.

In the Master-Slave model shown in Figure 2, same applications work on all of the PC nodes. The statuses of the applications are synchronized through a network, and the rendering regions thereof are suited to the creation of a seamless image on a large screen. *CAVELib* [VRC92] and *VRJuggler* [BJH*01] adopt this architecture for parallel rendering. In this Master-Slave model, it is of critical importance to achieve precise synchronization between all of the PC nodes so as not to incur a gap in image update timing. The synchronization is carried out with minimum network communication between the PC nodes. The amount of data communication is very small, as compared with that of the Client-Server model, and is not dependent on the target 3-D scenes. Therefore, this model is suitable for environments that have a relatively narrow-band network, such as a PC-Cluster using commodity network interfaces.

However, in the software environment based on the Master-Slave model, the synchronization mechanism is usually implemented manually by programmers. In *CAVELib* and *VRJuggler*, modifications of existent applications are necessary because these software environments, including the synchronization mechanism, are provided as libraries or frameworks that are presumed to be used by application developers. This means that we cannot fully inherit previous application resources because most of the source codes of the commercial applications that we use in daily life are not released. The cost of additional development using the libraries that are newly provided with the software environments is also considerable.

As an other absorbing approach, J. Jacobson et al. developed *CaveUT* [JL05] based on the commercialized *game engine* of *Unreal Tournament* [Epi04]. This approach cleverly exploits the game's open source code for supporting multi-projector environments. The modification of the source code is just few lines. However this method heavily depends on the *game engine* itself. When we want to use other *game engines*, the same approach is not guaranteed to be used on that *game engines*. In VR applications, many kinds of functions are requested to be used on multiple display systems. Hence we need to support various software environments including the *game engines* to provide appropriate functions for the requests.

3. Self-distributing Software Environment

In the section, we describe a self-distributing software environment that enables existent applications to be used on multi-projector displays without the need for special modifications. Our system is an attempt to achieve high transparency independent from the architecture of the PC-Cluster and the network bandwidth.

3.1. Basic Concept

In order to reduce the communication between PC nodes, which causes network bandwidth limitation, the proposed system adopts the Master-Slave model as parallel processing architecture. All of the slave nodes execute the same application, and they are well-synchronized via Ethernet, which is equipped in each PC node.

In addition, the synchronization mechanism is not integrated into the application itself, but rather into the proposed software environment. In previous systems using the Master-Slave model, developers had to write special codes supporting synchronization between the PC nodes. However, in the proposed system, which provides synchronization functions to the application, without the need for any special modifications, the extra cost of the additional development is reduced, and most applications, the source codes of which are not released, are also available.

3.2. API Interception

In order to apply the synchronization mechanism to the existent stand-alone applications, we exploit the characteristics of Application Programming Interfaces (APIs), which interface the application with its environment, including a system kernel, some libraries and the input devices. In general applications, special functions, such as accelerated 3-D graphics rendering, disk I/O, and system event handling, are called through the APIs provided by the software environment. When an application calls some functions through the APIs, the substance of the APIs is actually executed. These important APIs are usually released as a dynamic link library. Hence, we can execute arbitrary functions by replacing either the reference table of the API or the API itself.

In the proposed system, the functions including the synchronization, which are required for parallel image generation on the PC-Cluster, are applied by replacing the import-table of the target APIs. This application module, which can achieve the API interception, is called the "API adapter". Applying the API adapter to the existent applications, the proposed system adapts the existent applications automatically to the PC-Cluster-based parallel image generation without the need for special modification by the user. The working process of the API adapter is shown in Figure 3.

The architecture of the proposed system is illustrated in Figure 4. Each node has the same application and API adapter, and they are executed simultaneously. In the execution process, the API adapter stealthily builds a communication layer on the network and achieves synchronization between the applications working on all of the PC nodes.

3.3. Synchronization via API

In general applications, the results of the exchange toward the outside, such as user I/O or system event handling, decide the only status of applications. These communications

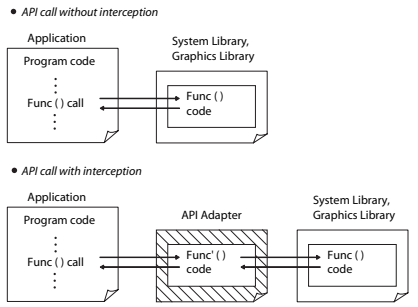

Figure 3: *Intercepting API calls.*

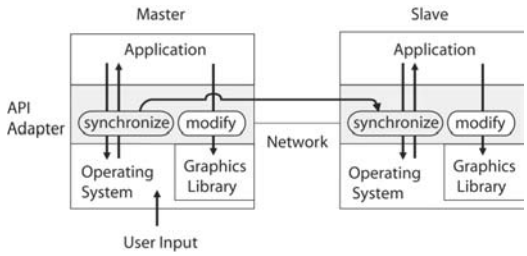

Figure 4: *Software architecture.*

are processed through the APIs. Hence, we can achieve system-level synchronization by applying the API adapter to APIs of which the results affect the execution statuses.

For example, in the execution of the same applications on each of the PC nodes without synchronization, the return value of the API may not correspond with that of the other PCs. A typical example is the time-dependent API, such as a high-precision timer, the results of which depend heavily on the CPU clock counter. The order and timing of system events are also different in each of the PC nodes. Therefore, the results of some processes that are controlled by event-driven architecture cause inconsistency of the execution status, as shown in Figure 5. These inconsistencies appear as differences in rendered images and degrade the performance of immersive projection displays.

In contrast, using the API adapter, the results of some impact APIs are well-synchronized, and the order and timing of the system events are also harmonized, as shown in Figure 6.

3.4. Supporting Multi-Projector Architecture

In a standard immersive projection display with a PC-Cluster, the full screen area is divided into a number of rectangular regions that are related to each PC node of the PC-cluster. In order to achieve seamless image projection on large screens, each of the PCs must render the images for the appropriate region synchronously. In the proposed system, by intercepting 3-D graphics APIs, appropriate view parameters for each of the PCs are set according to their lay-

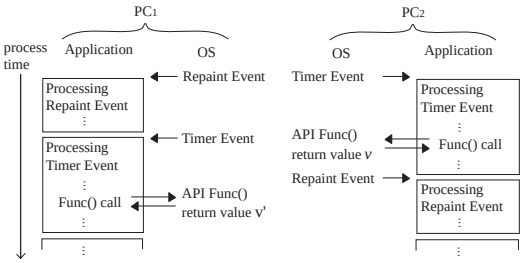

Figure 5: *Process flow without API adapter.*

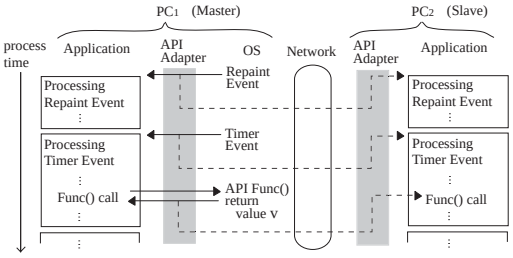

Figure 6: *Process flow with API adapter.*

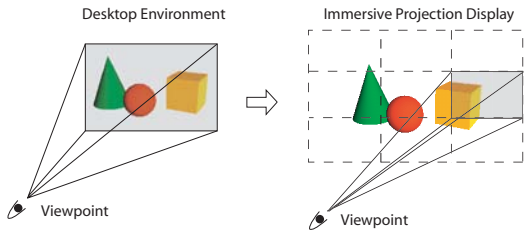

Figure 7: *View parameter modification.*

out, as shown in Figure 7. Preparing various view parameters beforehand enables the proposed system to support several kinds of display layouts.

However, in this strategy, the time required for rendering is different for each PC node because the rendered contents are decided based on the view parameters. Therefore, a *SwapLock* mechanism is crucial to synchronizing the update timing of the rendered images. Although the gap of the image update timing has no effect on the change of the execution status of the application itself, this is a difficult problem for users of multi-projector displays. Therefore, in the proposed system, *SwapLock* is also implemented by intercepting and synchronizing 3-D graphics APIs related to the update of output images.

Although some customized video cards support *SwapLock* and *SyncLock* needed for signal-level synchronization, such special hardware is not accessible for a commodity PC-Cluster. The particularity also spoils the PC's merit that can rapidly introduce latest hardware including a video card. Supporting *SwapLock* by software as our approach is significant to make the multi-projector displays be widely and easily used in various fields.

4. Implementation and Evaluation

In order to achieve highly realistic virtual environments, applications for the immersive projection displays are requested to process massive 3-D data in real-time. They are also required to achieve stable performance regardless of the rendering hardware so as to support various screen architectures.

Therefore, in this section, we describe the actual implementation of the proposed system on a multi-projector display "*D-vision*" [HJTS04], and evaluate the proposed system with respect to the two factors mentioned above. In order to investigate the availability on the commodity network interface, we use 100 Mbps *Ethernet*. *Chromium* [HHN*02], which is a typical implementation of the Client-Server model that can use existent applications without the need for special modifications by the user, was chosen as a target of comparison.

4.1. System Implementation

The system as implemented on *WindowsXP* and supported *OpenGL* applications is considered in this section. For system-level synchronization, we intercepted a handful of numerous system calls, which can alter application states. They include the calls to query Windows messages, the system timer and the input devices including a haptic interface. This system also supports multi-threaded programs if each of the threads works without affecting the internal stage of the program.

The multi-projector display *D-vision* used in this section is illustrated in Figure 8. In *D-vision*, a flat stereoscopic screen is used in the central area of the full screen in order to enable high-quality image projection. In the peripheral area of the screen, a simple curved screen constructed of fiberglass reinforced plastic (FRP) is used to achieve a wide viewing angle [YMT*02]. In this system, the entire image of rendered scenes is divided into 16 areas for distributed rendering by the individual PCs connected to each projector. Eight areas, including the central view and upper and lower viewing areas, are rendered by 16 PCs for stereoscopic viewing using polarized glasses. As a result, a total of 24 PCs are used for image generation using *D-vision*. The specifications of each PC node are shown in Table 1.

4.2. Synchronization procedure

In our implementation, we use "Detours" [HB99] for hooking APIs. It enables us to hook most APIs with same process.

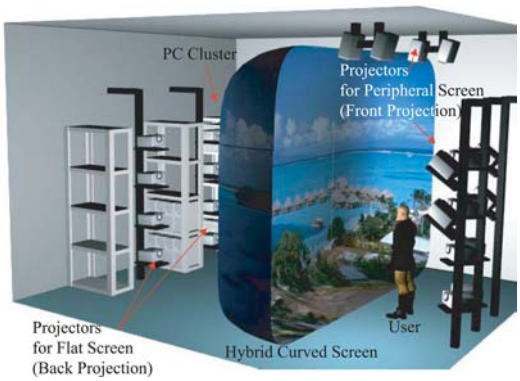

Figure 8: *Multi-projector display "D-vision".*

Table 1: *Specifications of PC nodes.*

CPU	Pentium4 2.4 GHz
Memory	512MB
OS	Windows XP HomeEdition
Graphics board	RADEON9700 Pro 128 MB

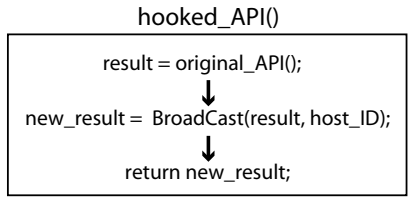

Figure 9: *Structure of hooked API.*

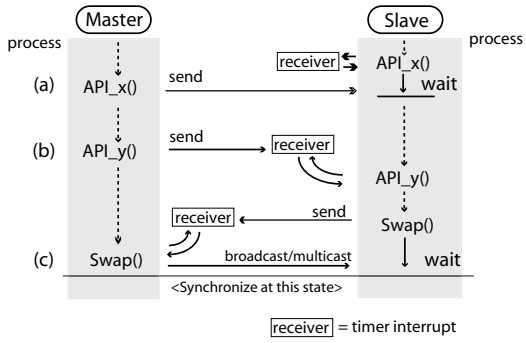

Figure 10: *Synchronization procedure between Master and Slave.*

In actual hooked API as shown in Figure 9, the original API is called as a first step. Next the result of the master node is broadcasted to all of the PC nodes. This broadcast means that all PCs share the result of the original API executed on the master node. Finally, the API returns the shared results to applications. This whole process achieves the synchronization of the result of the APIs. All of the communication is based on UDP protocol. The overhead of this process depends on the structure of applications and those execution states. In our trials, the overhead is almost within 0.5ms.

The APIs acquiring the system-dependent information like timeGetTime() and QueryPerformanceCounter() are necessary to be hooked, and its results must be shared with all of the PC nodes. The event messages sent to each process also affect the execution states. Therefore the APIs related to the message handling like a PeekMessage() have to be hooked. In multi-threaded processes, we have to manage the state of each thread. By hooking APIs generating threads like a CreateThread(), we can assign an independent communication port for thread-level synchronization. Because the number of these APIs is limited, this hooking process is the practical approach to synchronize the working processes on different PC nodes.

The synchronization procedure is shown in Figure 10. In Figure 10, Master node and Slave node are working with synchronization. The API_x(), API_y() and Swap() are hooked APIs. The execution states are synchronized, and same APIs are called in same order. A timer interruption function named "receiver" is used for the polling of the UDP-based communication. At first, in Figure 10(a), API_x() is called first at Slave node. Slave node waits for the communication from Master node. At Master node, the results are immediately sent to Slave node, and without waiting time, the execution process is continued. After that in Slave node, the result from Master node is received, and the paused process is resumed. Next, in Figure 10(b), API_y() is called first at Master node. Master node sends the results without waiting. At that time, the process of Slave node does not reach API_y(), so the receiver receives the communication instead. API_y() called at Slave node refers the receiver and acquires the results from Master node. In this process, there is no waiting time for Slave node.

Although these procedures are asynchronous at the time axis, the execution states are synchronized with sharing the API results from Master node. Needless to say, some APIs require synchronization at timing-level. For example, glxSwapBuffers() have to be processed at the same time in all PCs for seamless image projection. In our implementation, these kinds of APIs adopt more complex communication procedure different from the method described above.

In Figure 10(c), Slave node sent a message to inform Master node about the achievement to the Swap(). After the receipt from Slave node, Master node send a trigger message to all Slave nodes. In Figure 10(c), Slave node have to wait for the trigger message because Slave node's Swap() is called first. If Master node's Swap() is called first, Master node have to wait for the achievement message from Slave node. This trigger message releases so-called "SwapLock". After this process, all PCs are synchronized at timing-level.

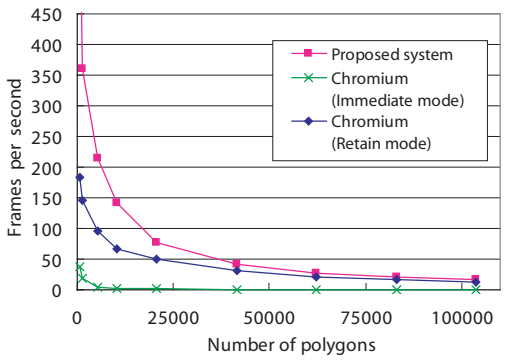

Figure 11: *Number of polygons and frame-rate.*

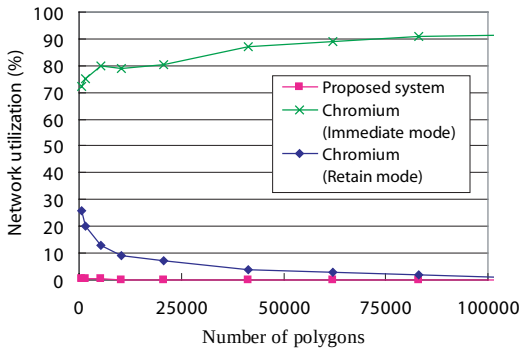

Figure 12: *Number of polygons and network utilization ratio.*

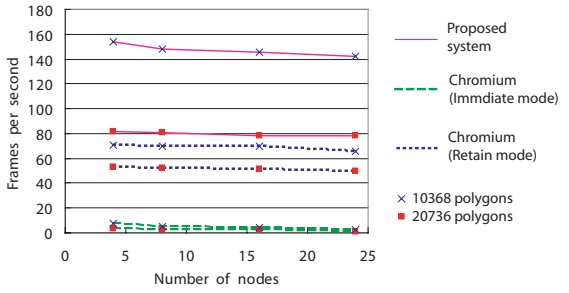

Figure 13: *Number of nodes and frame rate.*

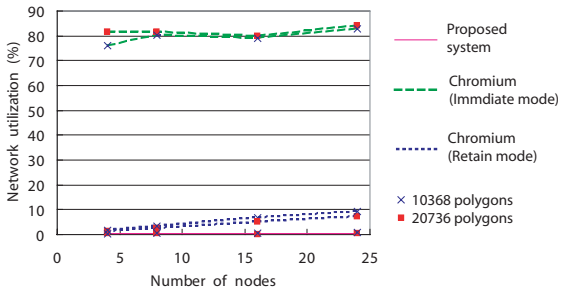

Figure 14: *Number of nodes and network utilization ratio.*

As shown in above, categorizing target APIs into two groups, one requires only order-level synchronization and another requires both order-level and time-level synchronization, greatly contributes to reduce communication delay.

4.3. Number of Polygons vs. Performance

First, we evaluated the rendering performance with respect to changes in the amount of target 3-D data. As a target application, we prepared a simple *OpenGL* application using *GLUT* [Kil98]. This application can freely control the number of polygons than must be rendered. In addition, this application involves no interaction with users during the evaluation process.

In this evaluation, we focused on the rendering modes of *OpenGL*: an immediate mode and a retain mode. While the immediate mode processes issued rendering commands immediately, the retain mode caches the commands beforehand and then calls them upon actual rendering requests. The immediate mode is a multipurpose mode because it imposes no restriction on application architecture. Although the retain mode can achieve high rendering performance, the target 3-D contents must be fixed in advance, as in a 3-D model viewer. We examined the adaptability of the proposed system to both the rendering modes. The results are shown in Figure 11 and Figure 12. Figure 11 indicates the rendering performance by the frame-rate (frames per second: fps) for the change in the amount of 3-D data, represented by the number of polygons. The network utilization ratio, i.e. the ratio of the input/output traffic to the total network bandwidth, is also illustrated in Figure 12. In the proposed system, almost no different was observed between the result obtained with the immediate mode and that obtained with the retain mode. Therefore, we illustrate only the results obtained with the immediate mode, which is considered to be useful for several different kinds of applications.

In Figure 11 and Figure 12, the proposed system achieves better performance, as compared with *Chromium*, regardless of the number of polygons. For example, the proposed system performed rendering with over 40,000 polygons at over 40 fps, which is practically acceptable performance for general VR applications. Although the application used in the present evaluation has two synchronization processes per frame, the network utilization is approximately 0.65% at maximum, because synchronization requires a small amount

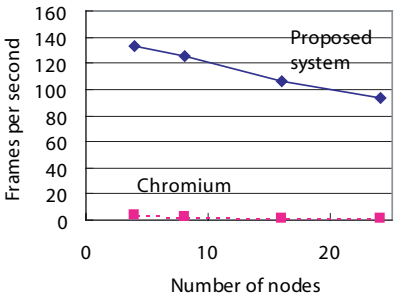

Figure 15: *Frame-rate for Quake III.*

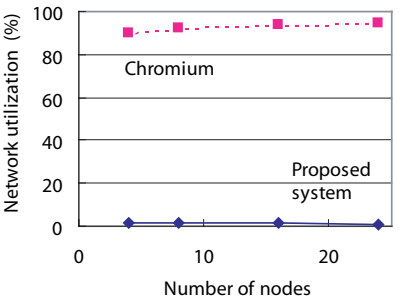

Figure 16: *Network utilization ratio for Quake III.*

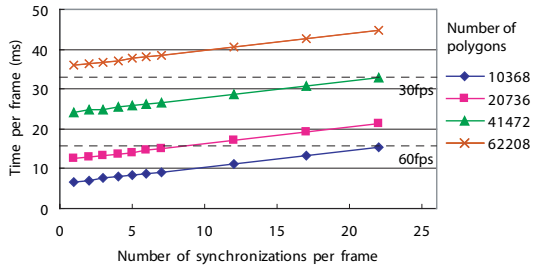

Figure 17: *Number of synchronizations and frame-rate (number of nodes = 24).*

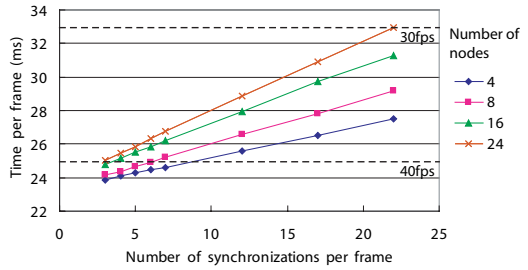

Figure 18: *Number of synchronizations and frame-rate (number of polygons = 41,472).*

of data, the range of which is from four to a few tens of bytes per process.

In contrast, *Chromium* gave different results in the immediate mode and the retain mode. Although *Chromium* achieved approximately 20 fps for 1,500 polygons in the immediate mode, the performance for over 5,000 polygons dropped to 5 fps or less. This is because the numerous communication data, which increase according to the target polygons, must be sent sequentially using the limited network bandwidth. This was also indicated by the high network utilization ratio, which was over 80%, as shown in Figure 12. However in the retain mode, rendering commands are cached at rendering servers, and the amount of data sent to the servers is vastly reduced. As such, high rendering performance is achieved by the reduced network utilization. From the results shown in Figure 11, *Chromium* performed at 36 fps and over with 40,000 polygons or less, which is sufficient performance for practical use.

The immediate mode is more important than the retain mode because it can be widely used in several kinds of applications. Therefore, the application fields of *Chromium* are restricted because of its insufficient performance in the immediate mode. In addition, in the retain mode, *Chromium* had worse performance than the proposed system with 50,000 polygons or less, because of the overhead of the compress and decompress phases in the communication protocol used in *Chromium*. Although these phases reduce the amount of data actually transferred, they also cause new time-consuming processes. Especially in situations in which the rendering task is not so heavy, for example, within 50,000 polygons, such communication overhead is dominant with respect to the total performance. In other words, the efficient communication performance of the proposed method is suitable for such 3-D applications on limited network bandwidth.

4.4. Number of Nodes vs. Performance

Next, we compared the proposed system with *Chromium* on four different PC-Clusters, having 4, 8, 16 and 24 nodes, respectively. We measured the rendering performance (fps) and network utilization ratio with the same application used in Section 4.3. The results are shown in Figure 13 and Figure 14.

Based on these results, we determined that both performances are reduced according to the increase in the number of nodes. In multi-projector displays, a greater number of nodes is used in order to generate images for the peripheral areas surrounding the viewer. Therefore, we must consider that an increase in the number of nodes does not simply decrease the number of rendering tasks per node, as in the case of general parallel processing on a PC-Cluster.

In the proposed system, the time required for communi-

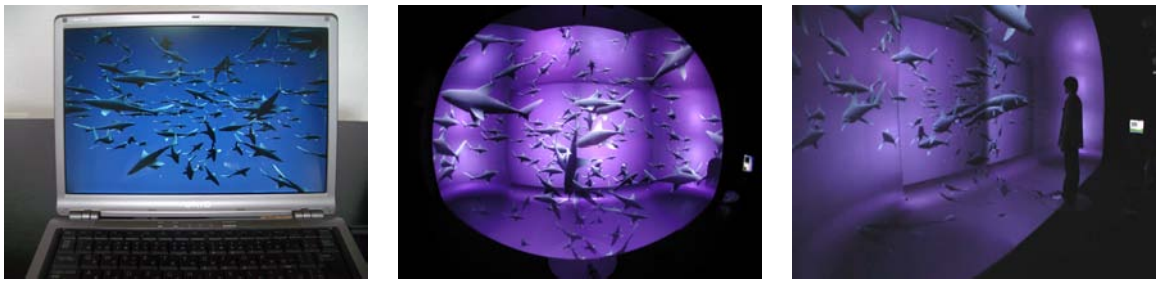

Figure 19: *In the proposed system, an application that was developed for stand-alone use (left image) can be used as-is on a multi-projector display. The center image shows the entire screen of D-vision executing the same application as that shown in the left image. Users can see the images in human-scale in the right image.*

Figure 20: *Complicated applications, such as commercial game software, are also adapted to multi-projector displays with a commodity PC-Cluster using 100 Mbps Ethernet. Applications projected in human-scale achieve novel interaction with users and helps to expand the fields of applicability.*

cation between the nodes is lengthened according to the increase in the number of nodes. However, the performance does not decrease in a linear manner because the communication processes are carried out in $O(log_2(n))$. Although the network utilization ratio with 24 nodes is also approximately 1.5 times that with four nodes, it is only approximately 0.6% of the entire network bandwidth. The proposed communication process does place some burden on the network bandwidth.

In multi-projector displays, increasing the number of nodes means increasing the screen area. As mentioned above, many more pixels have to be generated for high-resolution images. In the immediate mode, such a situation leads directly to a dramatic increase in network communications between each of the nodes. The time required for the communications dominates the overall process time as the network utilization reaches 80% or over for more than eight nodes. Therefore, the performance decreases relative to the increase in the number of nodes.

The performance of *Chromium* in the retain mode, which is robust to changes in the parallel rendering strategy, also decreases. It is because the increase of the nodes causes the increase of the communication frequency between the nodes. However, the rendering commands are stored at server nodes, and the network utilization is approximately 10% at maximum. Therefore, the network bandwidth can only be a bottleneck for a large-scale PC-Cluster.

5. *Game Engine*-based VR on Multi-Projector Display

Finally, in order to verify the capability of the proposed system for *game engine*-based applications, we executed commercial game software using both the proposed system and *Chromium*. Commercial game software called "*QuakeIII*" [Id 01], which is categorized as a first-person shooter, was selected as the target application. *QuakeIII* provides a realistic virtual environment with a high degree of interaction with the user, such as free movement using a joystick. As a recent challenge, the *game engine* of *QuakeIII* is applied to low-cost animation production, testbed for AI, interactive walkthrough for architecture, etc. This software

Figure 21: *QuakeIII's game engine is used for an entertainment application with human-scale interaction, and an architectural evaluation system. Human-scale locomotion and haptics interfaces are connected with API intercepting techniques. We can use haptic and simple gesture interaction with virtual characters (left image). 3-D contents are easily imported from other software like CAD (right image).*

has also been used in the evaluation of *Chromium*. Therefore, this software has typical characteristics of practical applications that may be used with multi-projector displays.

5.1. Performance on Scalable PC-Cluster

In our implementation on *D-vision*, *QuakeIII* could be executed in both environments without the need for additional modifications. Figure 19 shows the scalability of our approach, and Figure 20 shows *QuakeIII* as executed on *D-vision*, which has a surrounding screen. The frame-rate and network utilization ratio are also illustrated in Figure 15 and Figure 16. The target scene is constructed with approximately 10,000 polygons. For reference, a stand-alone PC node can render the target scene of *QuakeIII* at 156 fps.

Because *QuakeIII* used only the immediate mode, the network utilization ratio was extremely high in *Chromium*, as shown in Figure 16. As a result, *Chromium* achieved only approximately 4 fps, even for four nodes, having minimum data communication in this evaluation. That is to say, the user was not able to interact naturally with *QuakeIII*.

In the proposed system, the APIs that require synchronization were called approximately 15 times per frame. In total, 70% of the API calls was performed in order to obtain the precise system-time. In the proposed system, the bandwidth of the network did not become a bottleneck because the amount of communication data per synchronization was quite small. However, such successive synchronizations at approximately 15 times per frame causes considerable delay for real-time rendering processes. In the result for 24 nodes, which is the standard structure of *D-vision*, the total performance deteriorated by approximately 40%, as compared with the result obtained using a stand-alone PC.

In elaborate, high-quality applications, it is expected to use several APIs requiring synchronization. Therefore, we evaluated the performance according to the number of synchronizations per frame. For this additional evaluation, we prepared an application that can arbitrarily control the number of synchronizations, based on the application used in Section 4.3. The results are shown in Figure 17 and Figure 18.

The increase in the number of synchronizations leads to the increase in the amount of data transferred through the network. Therefore, the increase in the number of synchronizations loses the characteristics of the Master-Slave model, which requires low data communication between nodes. For *QuakeIII*, the proposed method achieved more than 90 fps, which is sufficient for practical performance, and there was no noticeable reduction in performance. However, based on the above results, the number of synchronizations greatly affects the execution performance. Therefore, the application fields of the proposed system must be examined carefully.

5.2. Examples of *Game Engine*-based Application

We used the *game engine*-based virtual environment for two practical applications. One is an entertainment application on *D-vision*. This application aims to achieve human-scale interaction with virtual objects and characters. We introduced a locomotion interface with walk-in-place motion and a human-scale haptic interface "*SPIDAR-H*" [HRJS04]. These devices are connected to the *game engine* by intercepting its various I/O APIs. In our implementation, we can use the haptic interface as a 3-D position tracker, and therefore simple gesture interaction is also accepted.

Another is an architectural evaluation system. Creating a virtual environment requires many techniques for its users. Therefore architectural researchers are not good at crystallizing their excellent ideas in interactive virtual environments. In this evaluation system, we used the *game engine* as an

easy authoring system of virtual environments. Architectural 3-D data designed with CAD is easily accepted. Human-scale interfaces are also available as same as the entertainment application.

The overviews of these trials are shown in Figure 21. Although almost same environments are provided with commercial software, *game engines* are extremely cost-effective, and they always provide newest technology with easy-to-use style.

6. Conclusions and Future Works

In this paper, we achieved a *game engine*-based virtual environment on multi-projector displays. In order to realize that, we developed a self-distributing software environment with API interception for system-level synchronization. This environment achieved low data communication based on the master-slave model. Applying this communication mechanism by intercepting significant APIs, the developed system executed existent *game engines* on multi-projector displays without the need for additional modification by the user. Based on evaluation results, we also clarified the characteristics of the proposed system through comparison to *Chromium*, a similar well-known system. Finally, we actually developed practical applications for entertainment and architectural evaluation with our proposed system.

As our future work, we have a plan to evaluate our proposed system with Gigabit Ethernet environment, widely spreading as a recent standard network interface of commodity PCs, and discuss about the effectiveness of that system as compared with previous approaches. We will also try to examine the possibility of *game engine*-based human-scale virtual environments by applying it to many kinds of fields.

References

[BJH*01] BIERBAUM A., JUST C., HARTLING P., MEINERT K., BAKER A., CRUZ-NEIRA C.: VR Juggler: A Virtual Platform for Virtual Reality Application Development. *Proc. of IEEE VR 2001* (2001), 89–96.

[CNSD93] CRUZ-NEIRA C., SANDIN D. J., DEFANTI T. A.: Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE. In *Proc. of SIGGRAPH '93* (1993), pp. 135–142.

[Epi04] EPIC GAMES INC.: Unreal Tournament 2004. `http://www.unrealtournament.com/` (2004).

[HB99] HUNT G., BRUBACHER D.: Detours: Binary interception of win32 functions. In *The 3rd USENIX Windows NT Symposium* (1999), pp. 135–143.

[HEB*01] HUMPHREYS G., ELDRIDGE M., BUCK I., STOLL G., EVERETT M., HANRAHAN P.: WireGL: A scalable graphics system for clusters. *Proc. of SIGGRAPH 2001* (2001), 129–140.

[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. *Proc. of SIGGRAPH 2002* (2002), 693–712.

[HJTS04] HASHIMOTO N., JEONG S., TAKEYAMA Y., SATO M.: Immersive Multi-Projector Display on Hybrid Screens with Human-Scale Haptic and Locomotion Interfaces. *Proc. of International Conference on CyberWorlds 2004* (2004), 361–368.

[HRJS04] HASHIMOTO N., RYU J., JEONG S., SATO M.: Human-Scale Interaction with a Multi-projector Display and Multimodal Interfaces. *Advances in Multimedia Information Proceedings - PCM2004 Part III, Springe* (2004), 22–30.

[Id 01] ID SOFTWARE INC.: QuakeIII Arena. `http://www.idsoftware.com/games/quake/quake3-arena/` (2001).

[JL05] JACOBSON J., LEWIS M.: Game Engine Virtual Reality with CaveUT. *IEEE Computer 38*, 5 (2005), 79–82.

[Kil98] KILGARD M. J.: The OpenGL Utility Toolkit Ver 3.7. `http://www.opengl.org/resources/libraries/glut/glut_downloads.html` (1998).

[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. *Proc. of IPT/EGVE 2003* (2003), 261–270.

[VRC92] VRCO INC.: CAVELib. `http://www.vrco.com/CAVELib/OverviewCAVELib.html` (1992).

[YMT*02] YAMASAKI M., MINAKAWA T., TAKEDA H., HASEGAWA S., SATO M.: Technology for Seamless Multi-Projection onto a Hybrid Screen Composed of Differently Shaped Surface Elements. In *Proc. Seventh Annual Immersive Projection Technology symposium* (2002).