ICAT 2005
December 5-8, Christchurch, New Zealand

VR Content Platform for Multi-Projection Displays with Realtime Image Adjustment

Takafumi Koike **Kei Utsugi** **Michio Oikawa**
Systems Development Laboratory, Hitachi, Ltd.,
1099 Ohzenji, Asao, Kawasaki, Kanagawa, 215-0013, Japan
{*koike,utsugi,oikawa*}*@sdl.hitachi.co.jp*

Abstract

We propose a VR content platform for multi-projection displays. Our platform has in common features with frame synchronization methods without any special hardware and real-time image adjustment mechanism with graphics processing units.

We realized the image adjustment mechanisms as image effects. The image effects used multipass rendering, multitexturing, and programmable shaders. Therefore we can easily extend adjustment algorithms, and use multiple effects, and implemented several image filters.

In above features, we can reduce workflow to create content for multi-projection display. Because we can efficiently operate post-productions for content for multi-projection display. We used our system to create high-resolution movie contents for seamless multi-projection display.

Keywords: Immersive Projection Technology (IPT), Frame Synchronization, Graphics Processing Unit (GPU), Realtime Image Adjustment, VR Content Creation

1. Introduction

Seamless multi-projection displays have been thoroughly studied and applied to many products, as one of the Immersive Projection Technologies (IPT). However, most of these products have simple architectures, for example some products consist of one PC and three displays. Therefore, studies on more complex systems are now in progress. For example, Yamasaki studied color and geometry adjustments for the overlapped areas using complex screen shapes and multi-projectors for immersive projection displays [12].

Additionally, special hardware is currently being studied for multi display systems. For the driver or library, Humphreys studied WireGL([4]) as a scalable OepnGL system on cluster PCs, and Chromium([5]) as a stream processing framework for interactive graphics on cluster PCs. For hardware, Stoll studied image reconstruction hardware for multi-projection displays [11].

However, the GPU processing power surpassed that of the CPU, and GPUs can execute most image-processing operations much faster than CPUs, [6] and [7].

In these displays, it is difficult to create motion pictures for immersive projection displays, because they have highly specialized, high-resolution screens. For example, during the editing of motion pictures, we have to execute fine color correction adjustments, and so on. But, during the adjustment, we must go back to the beginning to do some work, such as color correction (with tools such as Adobe® Photoshop® or After Effects®[1] on PC), encoding, and preview process in the actual displays. Chen studied scalable and high-resolution tiled displays([2]) by MPEG video delivery, but to use specific CODEC(COder/DECorder) is not suitable for VR contents creations. Because new codecs are developed continually, for example Pixlet for Quicktime 7 or WMV HD for WMV9 are these new codecs, and we want to choose codecs by creating contents.

To improve the workflow and solve these problems, we developed a synchronous GPU system over a network. Additionally, we applied the system to multi-projection displays to operate interactive and intuitive post-production of high-resolution motion pictures in realtime.

In section two, we show previos works and problems of past researches. In section threem we propose our architecture to rsolving some problems and show our experiment system. In section four, we show experiments results. Lastly, we conclude our research, and show future works.

2. Previous Work and Problems

2.1. Seamless Multi-Projection Displays

In our system, we used the techniques proposed by [12]. The system consisted of an equal number of PCs and projectors. Each PC generates and displays one image for one projector. In addition, each PC is connected over LAN (ex. 100/1000BASE-T), allowing us to realize fully scalable seamless multi-projection display systems, without restrictions on the number of projectors.

The images from each projector have overlapped areas. To realize seamless images, the test patterns were projected and measured using digital still cameras, and we calculated the correction data for seamless projections from the photos.

To correct images in realtime, we used specially developed image-processing boards. Each image-processing board had an image input and an image output. To realize seamless projection, these correction data were loaded to our boards, and corrective measures were applied to the input images.

After being geometrically transformed, the real-time rendered images were projected onto a hemispherical screen using six projectors, and the brightness and color were adjusted to produce one seamless image on the screen. The transformation was executed on our proprietary image processors.

It is not possible for this system to process image corrections using a GPU, because our prototype GPU-based image processing system runs at about 25 frames per second only, without any other graphic processing in Radeon 9800. This is because the image processing of our seamless

[1] Adobe, Photoshop, and After Effects are the registered trademarks of Adobe Systems Incorporated.

projection system processes geometry, color, and gamma corrections for each pixel, which require a huge number of texture lookups.

The hardware can process images in real time, but is not programmable, and the parameters cannot be uploaded in real time. Therefore, it is difficult to change the correction data in real time.

2.2. Frame Synchronization Problem

For simplicity, we used only movie content in our research, but we can easily extend this to real time VR content. In our system, we choose a Direct Show Technologies on Microsoft Windows platform because it supports many codecs, and we can play movies free from codecs. This enables us to choose the most suitable format for the content, depending on the circumstances. Additionally, developing codec is extremely difficult, and therefore impractical for our system.

We showed a movie profile (Figure 1) with a setting of 30 frames per seconds. The movie contained 2218 frames. The X-axis represents the frame numbers of the movie, and y-axis represents the interval time of the previous frame in milliseconds. This graph shows that there is a lot of movie playback jitter.

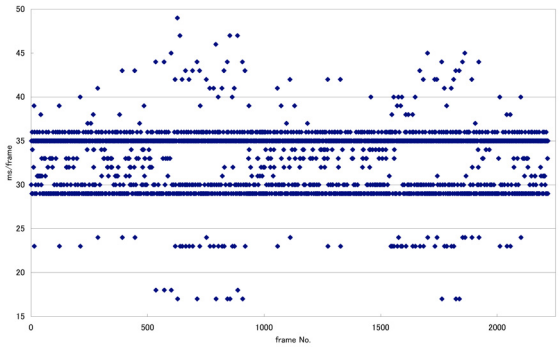

Figure 1: Movie profile

Next, a movie histogram is shown in Figure 2. The X-axis represents the interval time of the previous frame in milliseconds, and the y-axis represents the number of frames during this interval. This verifies that almost all frames were in time, and only a few frames were delayed.

Therefore, the above graphs indicate that the same start time alone is not enough for seamless multi-projection. If the movie requires camera panning, the difference in one frame can be very noticeable.

On the other hand, the PC clocks were incorrect, because their crystal oscillators are inaccurate. Therefore, if we use these systems for a long time, we need to stabilize the time synchronous logic. To realize this, it would be ideal to use special synchronization hardware, but too expensive. Therefore, we chose an approach that does not require special hardware.

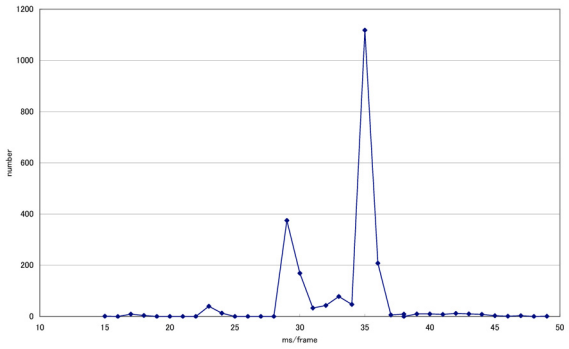

Figure 2: Movie histogram

2.3. Workflow of Contenet Creationfor Multi-Projection displays

We considered how to create content for multi-projection displays. Figure 3 shows the content creation workflow for multi-projection displays.

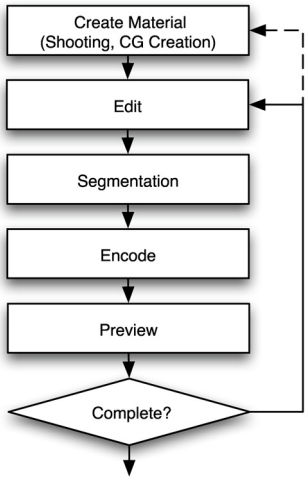

Figure 3: Content creation workflow for multi-projection display

At first, we created some content materials. For example, we created pre-rendered CG, or a title image, and so on. Some movies need fish-eye lenses, HD cameras, or multi-cameras. Of course, the resolution of pre-rendered CG is larger than that for one display. Our system has a resolution of 2576×1350, which is larger than the 1080p(1920×1080) HD format.

Next, these materials were edited using video editing tools, such as Adobe After Effects® or Apple Fincl Cut Pro®. We assumed a spherical display to create content. Usually, a flat panel display with a smaller resolution than that of multi-projection displays is used to create content, and its display forms are different from real environments (in this case, we used a spherical display). In addition, to

realize seamless images, we must sacrifice color correctness or brightness. Therefore, we cannot use the color-matching processes usually used to verify the scene. Therefore more previews and editing are needed.

Next, we must segment the contents. Segmentation is an accepted operation for seamless multi-displays because, to realize seamless images, they must be divided for each projector with some overlapped areas.

Consequently, encoding is also critical to multi-display systems, because very large image sizes do not allow movie playback in realtime.

Lastly, we preview the content, and if any corrections are needed, we must go back to edit the parts.

3. Methods

3.1. System Architecture

Figure 4 shows our proposed archtecture. We need at least two PCs, one is the controller PC and the other is the display PC. One display PC is required for each display/projector, for example, we require three display PCs if we have three display systems.

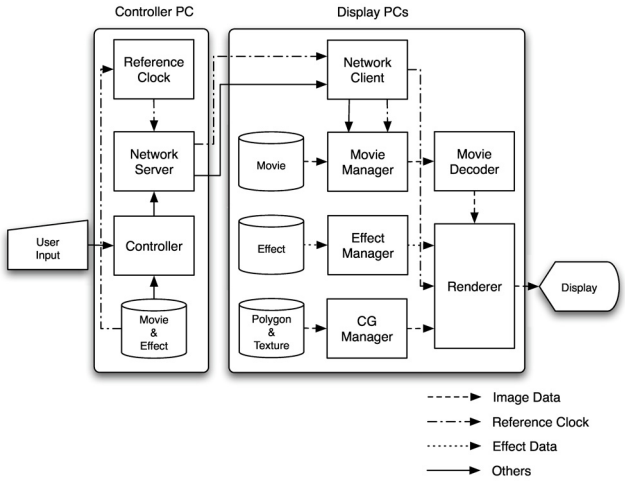

Figure 4: System architecture

The controller PC has three blocks. The first one is a controller block. The controller block treats the user interface and controls the other display PCs. The second one is a reference clock block. The reference clock block was used to generate our system's reference clock. If the movie has sound data, this block uses it because sound is very sensitive to speed playback differences. The reference clock has millisecond resolution. The third one is the network server block, which sends commands and the reference clock settings to the display PCs. Almost all commands are sent using TCP, but the reference clock is sent by UDP.

Display PCs have six blocks. The network client block receives various data from the controller PC via UDP and TCP, and sends appropriate data to the other blocks. The movie manager block reads the compressed movie data from local storage, and sends compressed movie data to a movie decoder block, with a reference clock. The movie decoder block decodes the compressed movie data and sends it to the GPU's texture memory. The effect manager block manages the previously described effect files, and sends them to the pixel or vertex shader of the GPU. The CG manager treats the polygon data and textures, and sends them to the vertex shader and texture memory, and, if needed, to the pixel shader. The rendering block is the main engine of the display PC, and it manages all the data from other blocks and, using the reference clock, renders movies and realtime CG.

To control the system, we prepared a GUI to operate the content playback and realtime image adjustments. The percentage and these adjustment parameters are broadcast to the other PCs. In addition, we developed an effect time line (ETL) format, based on XML, to record the effect time lines operated by users with GUI.

3.2. Frame Synchronization

Our approach did not require special hardware; in other words, we used only Ethernet™[2]. In addition, for simplicity, we did not use any known quality of services (QoS) technologies.

To realize stable synchronous play back, we must consider two facts. First, we observed a frame difference. Second, frame synchronization must be continued for many hours. A long run time is required because movie editing is time consuming. However, computer times can vary by a few seconds a day. This means we need to synchronize the clocks on each computer.

To do this, we developed the following four features:

1. Synchronization with reference clock
   First, the control PC was connected to the other PCs using TCP. The control codes for start and stop are example by TCP. If all PCs are connected, the control PC broadcasts its own times to the display PCs to adjust them to 0 time.

2. Synchronization with sound data
   We defined the time code of the sound files as the reference clock. If the content had no sound data, we used synthesized sound data, for example Null MIDI files.

3. Autonomus clock
   If PCs do not have a reference clock, all display PCs run autonomously using their own clocks.

4. Synchronization by texture
   In DirectShow technology, DirectShow plays movies using their own clocks, and there are limitations on synchronizing the clocks to the reference clock. To resolve this problem, we first decompress the cached movie data to the GPU's texture memory; then, to time the vertical sync signal, we controlled the frame synchronizations. When restricted to movie content, we used movie profile for synchronization, which profile is playback length of each frames.

3.3. Theory of Realtime Image Effects

First, we defined the image effects as follows. We defined position $\mathbf{x}$ in a space domain, and time t in a time domain. We only considered the image effects of 2d space for simplicity, but we can easily expand into 3d space. In 2d space, position $\mathbf{x}$ is expressed as $\mathbf{x} = (x, y)$.

[2]Ethernet is a trademark of Xerox Corporation.

Now we can define the image processing filters to project from $p(\mathbf{x}, t)$ to $q(\mathbf{x}, t)$.

$$q(\mathbf{x}, t) = k_d^{-1}(\mathbf{x}, t) \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\mathbf{x}', t', \mathbf{x}, t)p(\mathbf{x}', t')dx'dy'dt' \tag{1}$$

We defined $k_d(\mathbf{x}, t)$ as a normalized parameter as follows:

$$k_d(\mathbf{x}, t) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\mathbf{x}', t', \mathbf{x}, t)dx'dy'dt' \tag{2}$$

In this system, $q(\mathbf{x}, t)$ and $p(\mathbf{x}, t)$ usually had RGB pixels, and these vectors had three components as follows:

$$q(\mathbf{x}, t) = (R, G, B) \tag{3}$$

Some image processing filters use the differentials between adjacent pixels. In this case, we could easily realize these filters to treat $f(\mathbf{x}', t', \mathbf{x}, t)$ as a diffrential operators.

In addition, calculations have finite integral intervals in most cases, and because images are constructed using pixels, the calculations have discrete values, and the integrals are described by sums. In addition, in most cases, a was the small integer, and $[t' = t], [x' = x - a, x + a]$.

If the integral intervals were wide, the following is true.

$$\begin{aligned} q(\mathbf{x}, t) &= k_d^{-1}(\mathbf{x}, t) \times \\ &\quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{x-a}^{x+a} f(\mathbf{x}', t', \mathbf{x}, t)p(\mathbf{x}', t')dx'dy'dt' \\ &= k_d^{-1}(\mathbf{x}, t) \times \\ &\quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{x-a}^{x} f(\mathbf{x}', t', \mathbf{x}, t)p(\mathbf{x}', t')dx'dy'dt' \\ &\quad + k_d^{-1}(\mathbf{x}, t) \times \\ &\quad \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{x}^{x+a} f(\mathbf{x}', t', \mathbf{x}, t)p(\mathbf{x}', t')dx'dy'dt' \end{aligned} \tag{4}$$

We divided the two sums, to verify that they could be applied using multiple-filters.

3.4. Implementation of Realtime Image Effects

Movie textures are used to realize realtime image adjustments, and our system can play these movie textures synchronously, using the same time code as the motion pictures. A pixel shader was used to write the image process functions, and all images were treated as textures and rendered as square.

We added some adjustment functions, such as color curves, hue adjustment, mosaics, sepia, reversal, and edge-detect. Thus, editors can reuse these functions many times using multipass and multi-texture rendering.

All functions were implemented using HLSL, which allows us to easily add more functions.

For flexible movie playback, we used a Microsoft Windows platform, and Direct Show technologies. We used a VMR9, which rendered a D3D texture surface that could easily be reused for other 3D applications.

To realize the previously defined image effects, we developed a design with following five features.

1. Time-Line Filter
   In some types of filters, such as optical flow [1], we need to calculate time integrals, and to do this we used mutitexturing. We stored the frames in texture memory, and integrated them by adding the stored textures. If we store 1024×768 images in a 256-MB graphics board, each frame needs 3 MB of memory space. Therefore, the cards can store more than 80 frames.

2. Multiple Filter
   As mentioned, we considered the filters as integrals for space and time. We used a pixel shader to integrate the functions. If the integral interval is local, it could be calculated using texture lookups. However, both the operation numbers of the pixel shader and the number of texture lookups would be restricted. Therefore, if the integral intervals are wide, they are difficult to calculate. To avoid this problem, we developed a multiple filter mechanism. As shown in equation (4), we divided the integral intervals to fit the GPU hardware restrictions and then calculated the integrals. To realize multiple filters, we used textures as the cache memory of the calculations. Using multipass rendering [3, 8], we enhanced the integral intervals,

3. Input from Ouside Data
   The system needs input from outside data to compose still images (ex. titles, background images). Moreover, we need to generate text data to obtain text-based mosaics or time consuming post-processes. To meet this requirement, we prepared a mechanism to input outside data as texture data.

4. Expansion of Shader Languages
   To realize scalable architecture in this system, we used high level shader languages (HLSL) in DirectX®[3] developed by Microsoft. We also expanded these languages to control the parameters. We defined the denomination rules as parameters in our system, and some names were used for handshakes between computers, as described below.

5. User Interface
   Some effects require not only applying or not applying but also dynamically changing the parameters. An example of such an effect is a tone curve (Figure 5).

3.5. Effects

We implemented 11 image filters to create effects for our systems; 1) Luminance transition (fade in, fade out), 2) Monochrome, 3) Sepia, 4) Mosaic, 5) Text based Mosaic (ASCII art), 6) Forsen, 7) Posterization, 8) Color space transition, 9) Fourier Translation, 10) Tone Curve, and 11) Gaussian filter. All these filters were implemented using a 2.0 pixel shader. However, it is possible to implement the system using pixel shaders of 1.1 or less. Of course, all filters run in realtime, and are applied to movies.

Next, we provide detailed descriptions of the image filters.

1. Luminace transition
   We use luminance transition filters when we start or stop movies, or transit scenes. Fade-in is a filter that

[3]DirectX is a registered trademark of Microsoft Corporation.

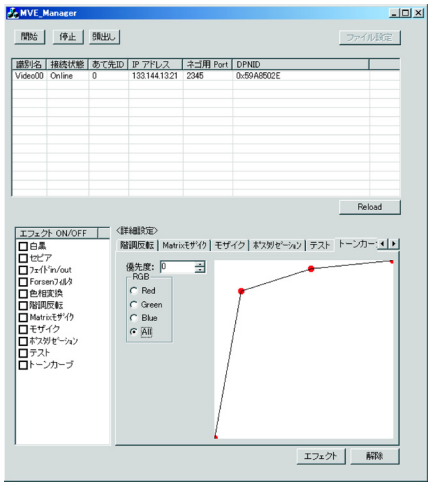

Figure 5: GUI for manipulating tone curve

gradually increases brightness, and fade-out is a filter that gradually decreases brightness. The only filter parameter is time.

2. Monochrome
Monochrome is a filter that converts color images to monochrome images. The only filter computation is luminance. Luminace(l) is calculated as follows.

$$l = 0.299 \times R + 0.587 \times G + 0.114 \times B$$

R and G and B are values of each color component.

3. Sepia
To realize a sepia filter, we first calculated the luminace of the images, such as the monochrome filter. Very like monochrome effect, we used 1d texture to realize.

4. Mosaic
The mosaic filter is the median of the squared areas. For simplicity, we calculated the median using the upper left of the squares. The width and height are controlable parameters. In spherical screens, we have to coorect the mosaic forms.

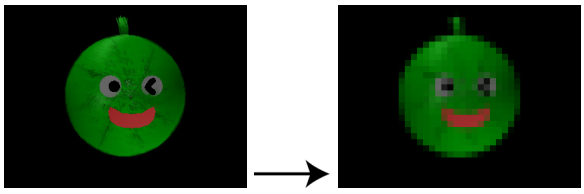

Figure 6: Mosaic Filter

5. Text-based Mosaic
Figure 7 shows the results of text based mosaic filter. First, we calculated the brightness, and allocated the pre-computed text required by specific areas. The text data are stored to 2D textures.

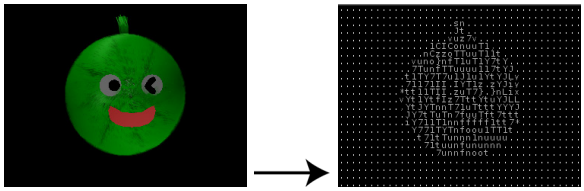

Figure 7: Text-based Mosaic Filters

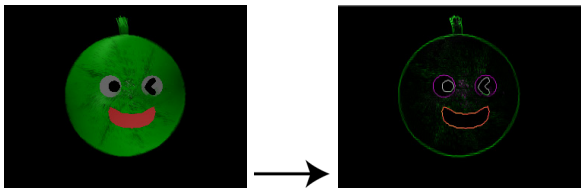

Figure 8: Forsen Filter

6. Forsen Filter
Figure 8 shows the results of this effect. The Roberts filter is known as a nonlinear difference filter which has edge-detection fanction [10]. The Forsen filter is known as a high-speed approximation of the Roberts filter [9].

7. Posterizaiton
Posterization is a filter that converts the colors of each pixels to colors within several threshold values.

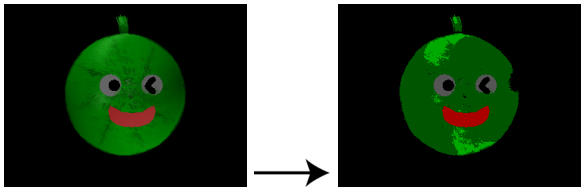

Figure 9: Posterization Filter

8. Color Space
First, we changed the color from RGB to HSV (Hue, Saturation, and Value) space. In HSV space, we rotated the color, and we changed it from HSV to RGB.

9. Fourier Transform
Several image filters, such as low-pass or high-pass filters, operate in a frequency domain. For these image filters, we implemented Fourier transform using fast Fourier transforms (FFT)

10. Gaussian Filter
Gaussian filters scumble the images. Figure 10 shows the results of this effect.

11. Tone Curve
The tone curve was the most effective filter used in this research. We applied color curves to each component, or all components with GUI (Figure 5). The color curve format is compatible with the Adobe ".acv" curve file, and we can import or export this format. Photoshop® can read and edit the formats.

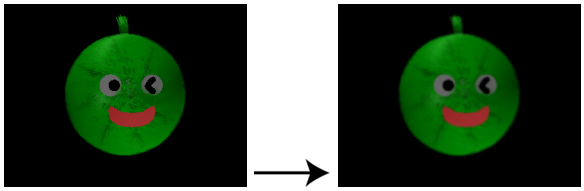

Figure 10: Gaussian Filter

In our system, we implemented only a minimum number of image filters, although the system has the architecture to easily add filters. In addition, this system has the ability to apply any of the filters in favorite order.

3.6. Workflow of proposed system

Figure 11 the proposed content creation workflow. This figure shows that editing might be required more than once, even though segmentation and encoding is required only once. Segmentation is particularly important for multi-displays. These two operations are necessary for computations and not for the editors. Further, by reducing the times we also reduce the costs. In this workflow, we tried to edit and preview more than in the old workflow.

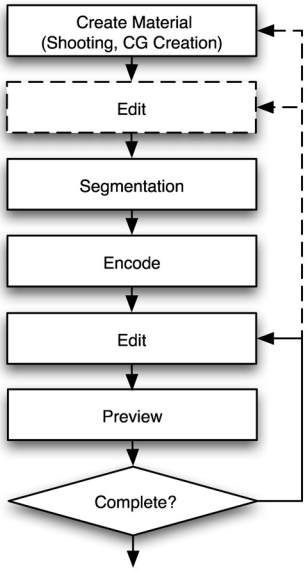

Figure 11: Content creation workflow for multi-projection display

4. Experiments and Results

We performed three experiments: 1) We measured the extent of synchronization; 2) We measured the performance of the multiple effects; and 3) We actually created a sample of the VR content, and checked the content creation workflow.

4.1. System Description

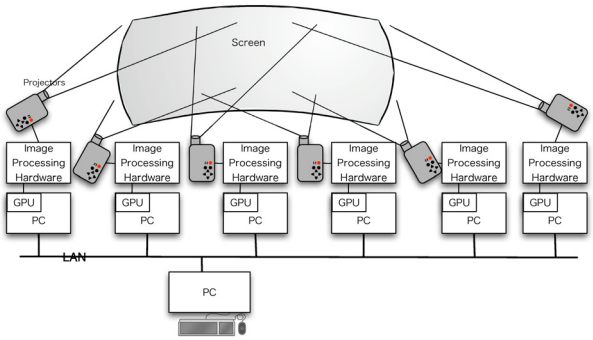

Figure 12: Experiment system

Figure 12 shows the architecture of our seamless multi-projection display system. The system consisted of 6 DLP^{TM4} projectors (1024×768 pixels) with front projection and has a hemispherical screen (curvature radius =2.1m). The viewing angle was 180 degrees. These projectors were arranged in a 3×2 array with a resolution of 2576×1350 because there were some overlapped areas (about 20% of each projector's images were overlapped). Each projector was connected to image processing hardware, whose sole purpose was to realize a seamless image. Image processing hardware had a video-input and a video-output connector that supports DVI. Each hardware unit was connected to a GPU, which was connected to the PC using an AGP bus. The specifications of each PC were as follows (CPU: Intel® Pentium® 4[5] 2.8 GHz, Memory: 1 GByte, HDD: 160 GByte, GPU: pixel shader 2.0).

This system was comprised of 7 PCs, one of which was the control PC and included a mouse, a keyboard, and a GUI display. Six PCs were controlled with one control PC over LAN (100BASE-T). In addition, the control PC could treat the movie sound files.

4.2. Frame Synchronization Performance

To confirm frame synchronization between each display, we used two methods: One was a physical eye check; and, the other check was a comparison of images shot with a DV camera. The movie that was used in experiments was a time code movie in MPEG1 format with 30 fps. Figure 13 shows the results of the experiment. Each of the four displays show the same frames. This proves that the synchronization difference was within one frame. Using eye checks alone, we could not confirm the synchronization differences. In addition, if our proposed method was not used, we confirmed the synchronization is unstable.

4.3. Performance of Multiple Effects

We performed 2 experiments to check the effects of the GPU and of image the filters. In the first experiment the CPU load showed no differences whether effects were applied or not. In the next experiment, five to six effects were achieved without frame drops and the CPU load did not increase.

[4]DLP is a trademark of Texas Instruments.

[5]Intel and Pentium are the registered trademarks of Intel Corporation.

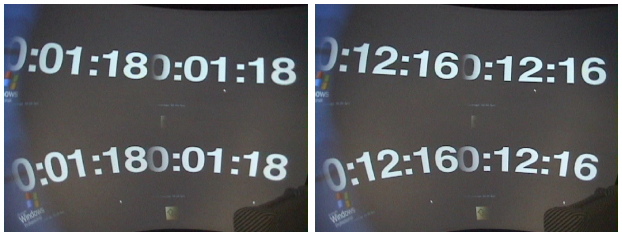

Figure 13: Results of Frame Synchronization(This photo shows 4 of 6 displays)

The top of Figure 14 shows an original movie we created. The bottom right of Figure 14 represents three filters; first the forsen filter, next the monochrome filter, and last, the tone curve filter was applied to the movie with 0 to 255, and 255 to 0.

4.4. Creating Sample Conetent

Figure 14 shows results of our system. Our original motion picture was rendered using a PC cluster that contained 20 CPUs over a period of about 4 days. The motion picture was divided into 6 movies per projector. The top image shows the original, the middle image was blued, and the bottom image was edge-detected and the image reversed. All the effects were controlled by users in real time, and all adjustment effects ran 30 fps without frame drops.

In this system, we improved the workflow and reduced the fine adjustment time for creating motion pictures using multi-projector displays.

5. Discussion

Above results of frame synchronization performance shows that software-based synchronizaiotn is sufficient. In multi-projection displays, it costs us by number of displays, so function replacements by software will decrease the costs of systems. The realtime image adjustments will also decrease the costs, and not only are image effects but also change VR content creation workflow.

Described above, our other research shows GPU based geometry and color correcntions for seamless multi-projection run about 25fps, so if we reduce functions or use future GPU, we will set up semless multi-projection displays without any special hardware.

We think clusters of next game consoles (ex. Xbox 360™[6], PLAYSTATION® 3[7]) have good cost performance, computation powers, and high-resolution image outputs, so these will be best platforms for seamless multi-projection displays.

In addition, these results suggest fully scalable systems for number of displays are aviable. We need fully scalable systems to realize stereoscopic displays with multi-viewpoint method. These autostereoscopic displays need more displays than that of stereoscopic displays and simple displays. Therefore our proposed techniques are useful for setting up large-scale stereoscopic displays with multi-viewpoint method.

[6]Xbox 360 is the trademark of Microsoft Corp.

[7]PLAYSTATION is the registered trademark of Sony Computer Entertainment Inc.

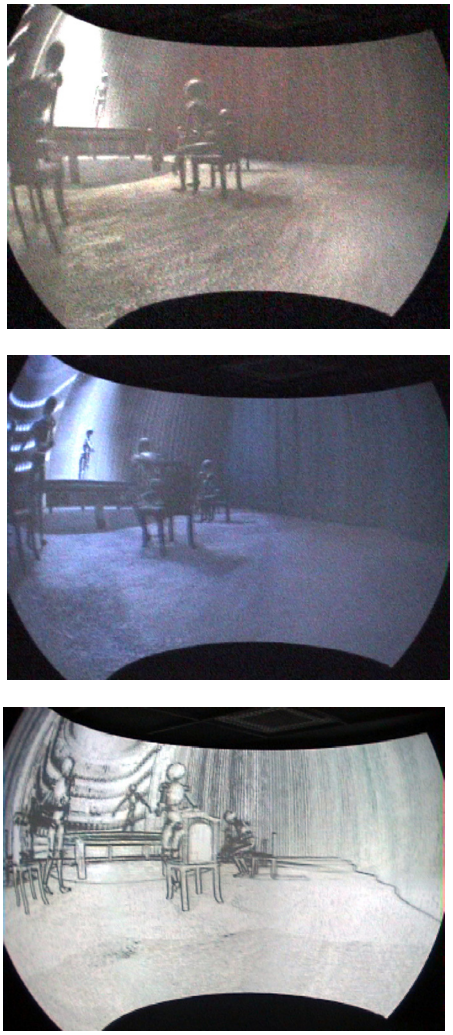

Figure 14: Image adjustment examples of this system

6. Conclusion and Future Work

We developed a VR content platform with multi-projection, frame synchronization without special hardware, and real-time image effects. The number of displays means that our system is scalable, and the image effects mean that it is extensible. In addition, we reduced time required for content creation and still obtained a high-resolution movie. Therefore, we proved our approach is adequate for VR systems with multi-projection displays. Our system also has an effective real-time image-processing feature.

In the future, because this system already has fade-in and fade-out support, if we add a support for playing more than one movie simultaneously, we can realize that most of editing part can be operated on multi-display system itself. This will allow us to create content efficiently.

Additionally, color changes can adapt the environment to suit the program by dynamically changing the surrounding lighting. Usually, VR systems are located in special places, such as laboratories and testing facilities. However, to spread multi-display systems throughout society, we must support dynamic light changing. By detecting dynamic environment changes, such as room brightness, the

system can change gamma or hue accordingly.

Next, this system will allow people to use 3D objects inside their computers, so they can easily apply real time CG For example, time changes can be easily expressed using tone curves, without any global illumination techniques.

Lastly, our system is suitable for camera array systems, and by creating special effects, the system might be capable of real time object recognition.

Acknowledgement

We would like to sincerely thank Takashi Ishijima for his advice about implementing this system, and Masami Yamasaki, Tsuyoshi Minakawa and Hideyuki Sakai for setting up the seamless multi-projection display.

References

[1] G. Adiv. Determining 3-d motion and structure from optical flow generated by several moving objects. *IEEE Trans. Pattern Anal. Machine Intell.*, pages 384–401, July 1985.

[2] Han Chen. "Scalable and Ultra-High Resolution MPEG Video Delivery on Tiled Displays". Technical Report TR-675-03, Princeton University Computer Science Department, 2003.

[3] Paul J. Diefenbach and Norman I. Badler. Multi-pass pipeline rendering: Realism for dynamic environments. In *Symposium on Interactive 3D Graphics*, pages 59–70, 1997.

[4] Greg Humphreys, Matthew Eldridge, Ian Buck, Gordon Stoll, Matthew Everett, and Pat Hanrahan. "WireGL: A Scalable Graphics System for Clusters". In *Proceedings of ACM SIGGRAPH 2001*, pages 129–140, 2001.

[5] Greg Humphreys, Mike Houston, Ren Ng, Randall Frank, Sean Ahern, Peter Kirchner, and Jim Klosowski. "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters". *ACM Transactions on Graphics*, 21(3):693–702, 2002.

[6] Jason L. Mitchell. Image processing with direct3d pixel shaders. In Wolfgang Engel, editor, *ShaderX: Vertex and Pixel Shaders Tips and Tricks*. Wordare, 2002.

[7] Jason L. Mitchell, Marwan Y. Ansari, and Evan Hart. Advanced image processing with directx 9 pixel shaders. In Wolfgang Engel, editor, *ShaderX 2 - Shader Tips and Tricks*. Wordare, 2003.

[8] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. Infinitereality: a real-time graphics system. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 293–302, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[9] W. K. Pratt. *Digital Image Processing*, pages 497 – 508. John Willey, 1991.

[10] L. G. Roberts. Machine perception of three - dimensional solids. In J. T. Tippett et al., editor, *Optical and Electro – Optical Information Processing*, pages 159–197. MIT Press, 1965.

[11] Gordon Stoll, Matthew Eldridge, Dan Patterson, Art Webb, Steven Berman, Richard Levy, Chris Caywood, Milton Taveira, Stephen Hunt, and Pat Hanrahan. "Lightning-2: A High-Performance Display Subsystem for PC Clusters". In *Proceedings of ACM SIGGRAPH 2001*, pages 141–148, 2001.

[12] M. Yamasaki, T. Minakawa, H. Takeda, S. Hasegawa, and M. Sato. "Technology for seamless multi-projection onto a hybrid screen composed of differently shaped surface elements". In *IPT 2002*, 2002.