# Distributed Rendering for Multiview Parallax Displays

Annen T.[a], Matusik W.[b], Pfister H.[b], Seidel H-P.[a], Zwicker M.[c]

[a]MPI, Saarbrücken, Germany
[b]MERL, Cambridge, MA, USA
[c]MIT, Cambridge, MA, USA

## ABSTRACT

3D display technology holds great promise for the future of television, virtual reality, entertainment, and visualization. Multiview parallax displays deliver stereoscopic views without glasses to arbitrary positions within the viewing zone. These systems must include a high-performance and scalable 3D rendering subsystem in order to generate multiple views at real-time frame rates. This paper describes a distributed rendering system for large-scale multiview parallax displays built with a network of PCs, commodity graphics accelerators, multiple projectors, and multiview screens. The main challenge is to render various perspective views of the scene and assign rendering tasks effectively. In this paper we investigate two different approaches: Optical multiplexing for lenticular screens and software multiplexing for parallax-barrier displays. We describe the construction of large-scale multi-projector 3D display systems using lenticular and parallax-barrier technology. We have developed different distributed rendering algorithms using the Chromium stream-processing framework and evaluate the trade-offs and performance bottlenecks. Our results show that Chromium is well suited for interactive rendering on multiview parallax displays.

**Keywords:** Multiview Parallax Displays, Distributed Rendering, Software/Optical Multiplexing, Automatic Calibration

## 1. INTRODUCTION

For more than a century, the display of true three-dimensional images has inspired the imagination and ingenuity of engineers and inventors. Ideally, such 3D displays provide stereopsis (i.e., binocular perception of depth), kineopsis (i.e., depth perception from motion parallax), and accommodation (i.e., depth perception through focusing). 3D displays that provide all of these depth cues are called *multiview autostereoscopic* or *automultiscopic* displays.[1] They allow uninhibited viewing (i.e., without glasses) of high-resolution stereoscopic images from arbitrary positions. Modern automultiscopic displays use either holographic, volumetric, or parallax technology.

Holographic displays record the complex amplitude (including phase) of optical wave fronts scattered off a three dimensional object. Consequently, the data rates of these displays are very large, and the display size is typically much smaller than average. Volumetric displays use rotating surfaces, transparent cubes of light-absorbing material, or stacked LCDs to isotropically emit light at points in 3D. These displays are typically not able to provide opacity, making the displayed images translucent, and none of them are able to reproduce view-dependent effects, such as reflections or specularity.

*Multiview parallax displays* are either based on parallax- barriers, lenticular sheets, or integral lens sheets. A parallax-barrier – introduced by Wheatstone in 1838 – is a raster-barrier placed in front of an image that blocks visibility of some parts of the image from each viewing angle. Lenticular or integral lens sheets consist of many micro-lenses which direct light from the image to different areas in the viewing zone. Lenticules are cylindrical lenses and provide only horizontal parallax, whereas integral lenses provide both horizontal and vertical parallax.

Further author information: (Send correspondence to Annen T.)
Annen T.: E-mail: tannen@mpi-inf.mpg.de, Telephone: +49 (0)681 9325 426
Matusik W.: E-mail: matusik@merl.com, Telephone: +1 617 621 7500
Pfister H.: E-mail: pfister@merl.com, Telephone: +1 617 621 7566
Seidel H-P.: E-mail: hpseidel@mpi-sb.mpg.de, Telephone: +49 (0)681 9325 400
Zwicker M.: E-mail: matthias@csail.mit.edu, Telephone: +1 617 452 2124

Multiview parallax displays offer many advantages that have lead to their dominance among recently introduced commercial 3D displays. They are automultiscopic without need for glasses or head-tracking technology. They can be manufactured easily by placing parallax-barriers or lens sheets in front of LCD and plasma panels without requiring prohibitive precision or cost. Because they are based on standard 2D display technology, they immediately benefit from economics of scale and future improvements in display resolution and quality. Multiview parallax displays are also capable of showing high-resolution 2D images, which is important for backwards compatibility and the display of text.

Multiview parallax displays project multiple views into the viewing zone. Current commercial flat-screen 3D displays are able to display up to nine perspective views,[2,3] whereas multi-projector 3D displays project 16 or more views.[4] This leads to a many-fold increase in image bandwidth. It also makes the design of the 3D rendering subsystem challenging, because multiview parallax displays must support interactive and immersive applications for entertainment (games), visualization, and virtual reality.

In this paper we present distributed rendering approaches for multiview parallax displays using multiple commodity graphics accelerator cards in PCs connected by a network. Our parallel rendering system is capable of driving multiview parallax displays with scalable rendering performance and resolution. We make the following contributions:

**Large-scale parallax-barrier display:** We built several large ($72'' \times 48''$) rear- and front-projection multiview parallax displays with 16 independent perspective views. We present background and a classification of projection-based parallax displays and a novel rear-projection parallax-barrier display prototype.

**Automatic calibration for parallax-barrier displays:** We present a novel, fully automatic camera based calibration procedure for parallax-barrier displays.

**Evaluation of distributed rendering strategies:** We describe various distributed rendering approaches for multiview parallax displays and evaluate their trade-offs and performance bottlenecks.

**Chromium-based implementation:** To make our rendering techniques widely available, we have implemented them non-invasively using Chromium.[5] Our methods can be applied to existing applications, such as Quake III, without modification or re-compilation.

**Practical demonstration:** We present results of interactive OpenGL applications running on our systems and describe the user experiences and avenues for future work.

## 2. PREVIOUS WORK

### 2.1. Multi-Projector 3D Displays

The literature on 3D display technology consists of thousands of publications and patents, see[6,7] for excellent overviews. Multiview parallax displays require an underlying display resolution that is the product of the resolution of each view and the number of views. To meet this substantial challenge for maximum HDTV output resolution with 16 distinct horizontal views requires $1920 \times 1080 \times 16$ or more than 33 million pixels, which is well beyond most current display technologies.

Fortunately, multi-projector display systems offer very high resolution, flexibility, excellent cost-performance, scalability, and large-area images.[8,9] The use projector arrays for 3D displays goes back to the beginning of last century.[10] More recently, Liao et al.[11] use a $3 \times 3$ projector array to produce an integral display with three views of horizontal and vertical parallax and an output resolution of $240 \times 180$ pixels.

The design of our multi-projector 3D displays was influenced by the 3D TV system of Matusik and Pfister.[4] We implemented their front- and rear-projection lenticular display with a linear array of 16 projectors. In addition, we also implemented a novel rear-projection parallax-barrier display using a $4 \times 4$ array of projectors (see Section 5). Our 3D displays have a display area of $72'' \times 48''$ and project 16 different perspective views at $1024 \times 768$ resolution into the viewing area. The large physical dimension of our displays lead to a very natural and immersive 3D experience.

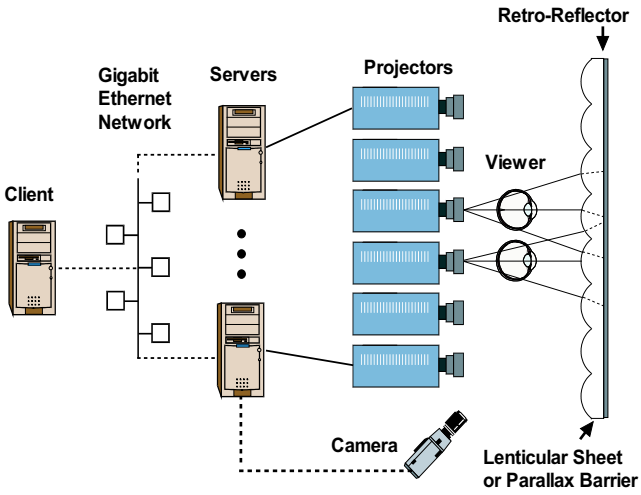



**Figure 1.** Our distributed rendering and display system.

Precise manual alignment of the projector array is tedious and becomes downright impossible for more than a handful of projectors. Similar to previous work[4, 8, 11, 12] we use a static camera for automatic geometric alignment of the projected images. We present a novel, fully-automatic geometric calibration method for the precise alignment of parallax-barrier displays in Section 5.

Of particular importance in multi-projector displays is to correct for the inherent differences in color and intensities between projectors.[13–15] We use the method by Majumder and Stevens[16] to correct photometric non-uniformities in our systems. We do not address the important issues of optical distortions[17] or interperspective aliasing[18] in multiview parallax displays.

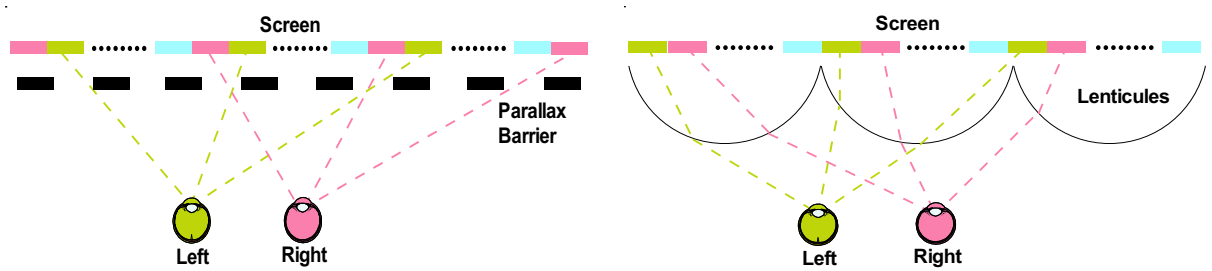
### 2.2. Cluster Graphics

Graphics rendering[19] and rendering for tiled displays[20] can be efficiently parallelized on clusters of PCs. Many techniques require modifications of application source code, which is often not available. Instead, we prefer non-invasive techniques for manipulating existing graphics applications. We use Chromium,[5] a modern stream processing API for clusters of PCs. Chromium's stream processing units (SPUs) are able to modify or replace any of the graphics API calls made by the application. In addition, many useful SPUs are available that can be adapted for our needs. We will discuss the details of our rendering approaches for multiview parallax displays in Section 6.

Opticality (formerly known as X3D Corp.), a commercial provider of 3D display technology, sells OpenGL Enhancer for non-invasive adaption of existing OpenGL applications for their multiview parallax displays. However, the tool does not support distributed rendering, and no publications with further details are available. Our proposed solutions are distributed, scalable, and provide interactive rendering rates for high-resolution 3D displays with 16 or more views.

## 3. SYSTEM OVERVIEW

Figure 1 shows a schematic representation of our overall system. One client PC and $n$ server PCs with fast graphics accelerator cards are connected by a gigabit ethernet network. The server PCs are connected to projectors (we use one projector per server) that display the appropriate views on the multiview parallax display. The figure shows a front-projection setup, although we also implemented rear-projection 3D displays (see Section 4). One of the servers is connected to a camera placed in the viewing area for automatic display calibration. After a general description of multiview parallax displays we describe the implementation of our 3D display prototypes in Section 5. In Section 6 we investigate the issues in constructing a high-performance rendering system driving those 3D displays. In Section 7 we present results from experiments using our distributed rendering approaches.




(a) Parallax-barrier displays (b) Lenticular displays

**Figure 2.** Multiview 3D displays with horizontal parallax using parralax-barriers 2(a) or lenticular sheets 2(b).

## 4. MULTIVIEW PARALLAX DISPLAYS

Multiview parallax displays rely either on *parallax-barriers* or *lenticular sheets* to multiplex several views into the viewing zone as illustrated in Figure 2. A parallax-barrier is a mask of parallel opaque stripes that reveals different parts of the underlying image depending on the viewing direction. The same effect can be achieved with lenticular sheets, which are linear arrays of narrow cylindrical lenses. Both techniques are used to show a single view from each position in the viewing zone. Hence, a viewer experiences parallax motion and binocular stereo without the use of special glasses. Different arrangements of lenticular sheets and parallax-barriers in multi-projector systems can be found in.[6] To project $k$ views at $m \times n$ resolution requires an underlying image with $km \times n$ pixels. We will refer to pixels of the underlying image as *sub-pixels* to distinguish them from the view-dependent *multiview pixels*, which comprise all corresponding sub-pixels visible through an individual parallax- barrier slit or cylindrical lense. Vertical slits or lenses result in a $k$-fold loss in horizontal resolution. Slanting the slits or lenses at a small angle[1] balances the loss of resolution in both directions. Parallax-barriers and lenticular sheets provide only horizontal parallax. Horizontal *and* vertical parallax is obtained using arrays of spherical lenses, or *integral lens sheets*. However, integral displays sacrifice significant spatial resolution in both dimensions to gain full parallax.

Important parameters of lenticular sheets and parallax-barriers are the number of lenticules (slits) per inch and the field-of-view (in degrees) of the viewing zone. While parallax-barriers reduce some of the brightness and sharpness of the image, lenticular sheets suffer from increased blurriness due to light diffusion inside the transparent sheet substrate. All multiview parallax displays suffer from crosstalk between neighboring images.

## 5. 3D DISPLAY PROTOTYPES

We implemented both a front-projection and a rear-projection 3D display prototype as illustrated in Figure 3(c and d). The front-projection system (Figure 3(a)) uses a lenticular sheet and a retro-reflective screen. The lenticular sheet both optically multiplexes and de-multiplexes the light. In the rear-projection system in Figure 3(b), each projector covers only a quadrilateral tile of the display surface. Tiling the display with an array of projectors allows us to display a very high-resolution image onto the diffuser. In contrast to the front projection system, the image tile of each projector does not only contain a single view, but multiplexes parts of all views. On the viewer side, we use a a lenticular sheet or a parallax-barrier to de-multiplex the light. Note that, in both cases, equivalent systems can be built using either parallax- barriers or lenticular screens. We call the view multiplexing in systems in Figure 3(a) *optical multiplexing* and refer to the view multiplexing for the rear-projection system in Figure 3(b) as *software multiplexing*. This distinction will be important when we discuss distributed rendering algorithms in Section 6.

Figure 3(c and d) show our prototype displays. We use 16 NEC LT-170 projectors with $1024 \times 768$ native output resolution. The front-projection lenticular display with optical multiplexing (Figure 3(c)) uses $72'' \times 48''$ lenticular sheets with 30 degrees field of view and $180/30 = 6$ viewing zones. The 16 projectors are arranged in a linear array. We tried to match the horizontal separations between cameras and projectors approximately, which required mounting the projectors in separate rows. For a more detailed description of identical 3D display systems see Matusik and Pfister.[4]

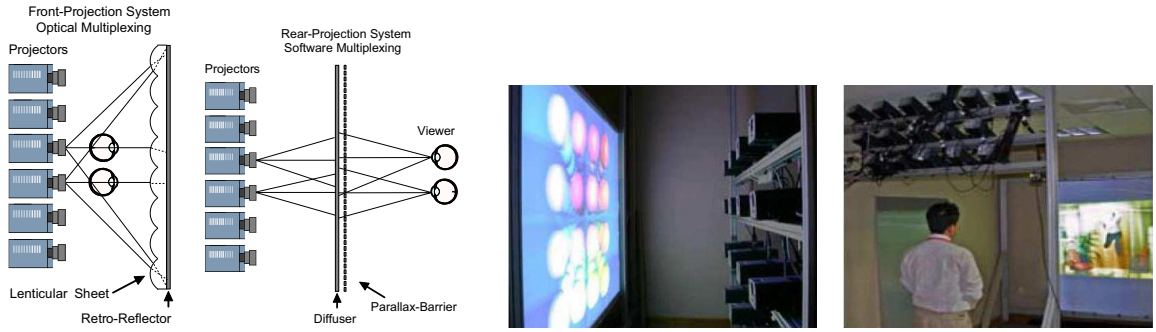
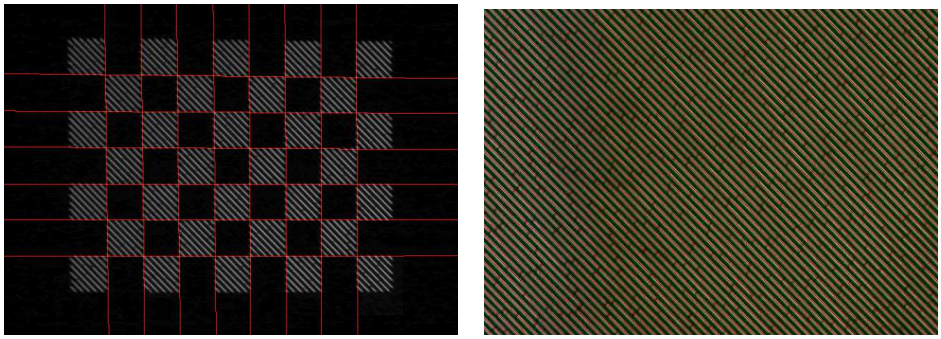



**Figure 3.** Rear-projection and front-projection multiview parallax displays setups are shown in 3(a) and 3(b). Equivalent systems can be built using parallax-barriers instead of lenticular sheets. 3(c) shows our front-projection lenticular display implementation with optical multiplexing and 3(d) shows our rear-projection parallax-barrier display with software multiplexing.

The rear-projection parallax-barrier display with software multiplexing (Figure 3(d)) is our latest system that has not been previously published. The projectors are arranged in a $4 \times 4$ array and project an image with $4096 \times 3072$ resolution onto a diffusing rear-projection screen. Instead of a lenticular sheet we produced a custom-made $72'' \times 48''$ parallax- barrier. We use a computer-guided laser to cut the parallax-barrier slits into a sheet of black Mylar polyester film. The Mylar sheet is glued to a sheet of glass that is mounted about 10 mm in front of the rear-projection screen. The slits are 1 pixel wide and spaced 16 pixels apart to project 16 views into the viewing area.

We have developed a novel procedure to automatically calibrate the position of the parallax-barrier with respect to the underlying rear-projection image. We place a highresolution ($3K \times 2K$) digital camera facing the center of the screen in the viewing area approximately 3 meters away from the screen. Our procedure has three steps: first, we compute homographies to register all projector images in a common coordinate system; second, we determine pixels in all projector images that are visible from one view-point; third, we compute pixels for other views using interpolation. We start by displaying a checkerboard images on each projector and taking corresponding images. We fit both horizontal and vertical lines to each of the observed checkerboard images as shown in Figure 4(a). Next, we compute intersection of each horizontal line with each vertical analytically. This gives us 48 coordinates of checker corners in each of the camera images. We can find the mapping of points on the display to camera pixels. This is expressed by a $3 \times 3$ homography matrix. In the next step, we project a white image on each projector and capture a single image with the camera. This creates a continuous bright surface to the human eye. However, the resolution of our camera is sufficiently high to detect the slits in the parallax-barrier. The captured camera image has high intensities in the slit areas and it has low intensities in the other areas of the parallax-barrier. First, we apply a Laplacian edge detector to the observed image, and we threshold the result. This operation produces pixels that belong to the slits. Next, we perform a flood-fill operation to group the pixels corresponding to each slit. Once we have all pixels that belong to the same slit grouped together, we use a linear regression to compute an analytical line equation for the slit. We show a closeup of the captured image and the superimposed analytically computed slits (shown in red) in Figure 4(b). The homographies allow us to transform the slit lines from camera space to the image space of each projector. In our setup, the slit lines in the projector images are roughly 16 pixels apart our display supports 16 distinct views. In order to obtain the slit lines for the other 15 views, we interpolate the line equations for the neighboring original slit lines. Once we rasterize all lines for all 16 views, we entirely fill the projector image space each sub-pixel in the projector space has a view assignment. This assignment is sufficient to perform the software multiplexing of views to sub-pixels.



(a) (b)

**Figure 4.** Calibration of the multi-projector parallax-barrier display. (a) Checkerboard and detected lines. (b) Maximum intensity image and detected slit lines.

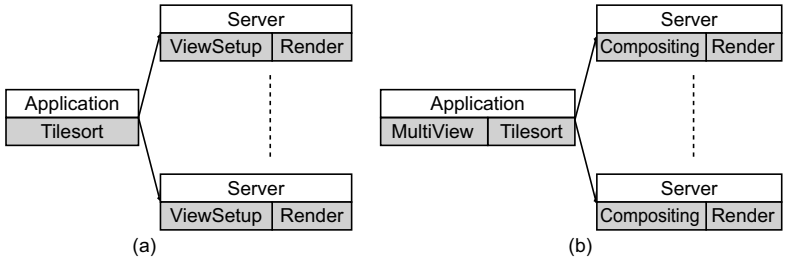



**Figure 5.** Chromium setup: (a) optical multiplexing, (b) software multiplexing.

## 6. RENDERING APPROACHES

We now present two strategies to render 3D scenes to a multiview parallax display and describe their implementation in a distributed rendering environment using Chromium.[5] We will exploit Chromium's concept of programmable *stream processing units* (SPUs), which act as filters on the stream of OpenGL commands. SPUs allow us to re-implement desired rendering commands and modify the way they are processed. Since Chromium does not allow us to define custom commands that can be inserted into a stream, we will use existing OpenGL commands to piggy-back information about special events that are necessary to control multiview rendering. Chromium SPUs can have internal state variables, which we will use to share information between the custom implementation of different commands. In the following sections, we will describe the SPUs that we have developed for multiview rendering.

As discussed in Section 4, in *optical multiplexing* (Section 6.1) each view is rendered as a whole and multiplexing of the views, i.e., the assignment of sub-pixels to multiview pixels, is achieved by the physical setup of the display. In contrast, in *software multiplexing* (Section 6.2) the rendering software combines all the views and assigns to each multiview pixel the corresponding sub-pixels.

### 6.1. Optical Multiplexing

In optical multiplexing, each view is projected as a whole to the display surface. After display calibration (see Section 5) we can ensure that each multiview pixel receives the contributions of the corresponding sub-pixels.[4]

Optical multiplexing is straightforward to implement in a distributed rendering environment; our Chromium setup is illustrated in Figure 5a. Each Chromium server is assigned to one view, i.e., the number of servers corresponds to the number of views. We use the standard Chromium `tilesort` SPU to distribute the rendering commands. Each server renders its view by piping the rendering commands through a custom `viewsetup` SPU followed by the standard `render` SPU. The `viewsetup` SPU intercepts commands changing the viewing matrix, modifying it to match the view associated with the server; the `render` SPU feeds all commands into a standard OpenGL pipeline for rendering. Each Chromium server is assigned to one view, i.e., the number of

**multiview SPU**

```
SPU multiview {
Matrix viewMatrix;

<-LoadMatrix(Matrix m) {
    m_prime = m.Premultiply(viewMatrix);
    ->LoadMatrix(m_prime);
  }

  <-SwapBuffers() {
    for(view=0; view<nViews; view++) {
      viewMatrix = computeViewMatrix(view);
      ->DepthBoundsEXT(view, nViews);
      ->playBack();
    }
    ->DepthBoundsExt(END_MULTIVIEW_FRAME, 0);
    ->SwapBuffers();
  }
}
```

**compositing SPU**

```
SPU compositing {
  bool beginFlag;

  <-DepthBoundsEXT(float zmin, float zmax) {
    if(zmin!=END_MULTIVIEW_FRAME) {
      beginFlag = true;
      ->renderDepthMask(zmin);
      ->DepthBoundsEXT(0.0, 0.5);
    } else {
      ->renderAlphaMask();
    }
  }

  <-glClear(enum mask) {
    if(beginFlag) {
      beginFlag = false;
      ->glClear(mask & ~DEPTH);
    } else {
      ->glClear(buffer & ~(COLOR|DEPTH));
  }
}
```

**Figure 6.** Pseudo-code for the new `multiview` and `compositing` SPU. The symbol `<-` indicates commands read from the previous SPU in the pipeline. Similarly, `->` indicates commands that are passed on to the next SPU.

In this scenario, the rendering command stream is sent just once per frame from the application to all servers, inducing little network bandwidth overhead for multiview rendering. However, the display setup required for optical multiplexing has a number of disadvantages. Since each projection needs to cover the entire display, the distance between projectors and the display surface is typically quite large, e.g., $2\sqrt{3}/3$ times the display width for a field of view of 60 degrees. Flat-screen displays, for example using high resolution LCDs, are not feasible with optical multiplexing. In addition, changing the field of view requires tedious readjustments of the projector positions.

### 6.2. Software Multiplexing

In software multiplexing, sub-pixels are projected directly to their corresponding multiview pixels by the rendering software. Hence, we can reduce the need for optical elements to a diffuser and a single lenticular sheet or parallax-barrier (see also Figure 3).

We implemented distributed rendering with software multiplexing using a tile-based architecture. Each Chromium server is assigned to a display tile and renders all the views for this tile. During rendering, the sub-pixels are interleaved such that they appear at the correct positions on the diffuser. The architecture of Chromium SPUs involved in this approach is shown in Figure 5b.

On the application side, our implementation consists of a custom `multiview` and a standard Chromium `tilesort` SPU. The `multiview` SPU transforms the application command stream for a single view of a frame into a stream that renders a *multiview* frame; pseudocode is given in Figure 6(left).

The SPU first packs all incoming rendering commands for one frame into a buffer. The `glSwapBuffers` command then triggers a loop over all views. For each view, the SPU sets up the appropriate viewing matrix. Then it inserts a special command into the stream to signal the servers which view is about to be transmitted. Next, the packed application rendering commands are replayed. After all views have been processed, another special command is added to the stream to indicate the end of the multiview frame. We chose to misuse the depth bounds test command `DepthBoundsEXT` and encode its meaning (beginning of a view, end of a multiview frame) in its *zmin* and *zmax* parameters. In case of conflicts with application commands, its role can be taken over by any other command that is not used by the application.

After the `multiview` SPU, the command stream is processed by a standard `tilesort` SPU that bucket-sorts geometry based on the view frustum of the tiles and the current view and sends the appropriate commands to

the corresponding servers. This culling mechanism effectively reduces the network bandwidth overhead due to the re-transmission of the rendering commands for each view.

On the server side, the command stream flows through a custom `compositing` SPU, followed by the Chromium `render` SPU. Pseudocode for the `compositing` SPU is shown in Figure 6(right).

First, this SPU re-implements the designated `DepthBoundsEXT` command and interprets its parameters to decide whether the command indicates the start of a view or the end of a multiview frame. If the beginning of the command stream for a view is detected, the SPU renders a depth mask that is used to discard all pixels in the tile that do not belong to the current view (encoded in the *zmin* parameter). For this purpose, the SPU enables the OpenGL extension `DepthBoundsEXT`. This allows masking pixels without interfering with depth, alpha, or stencil testing performed by the application. When the end of a multiview frame is detected (i.e., a `DepthBoundsEXT` command with appropriate parameters appears), the `compositing` SPU inserts commands to render an alpha mask into the stream, which will smoothly blend between tiles in the overlapping boundaries. Lastly, the `compositing` SPU also re-implements the `glClear` command. This command is usually issued by the application at the beginning of a frame. However, we must clear the framebuffer only at the beginning of each multiview frame; to preserve the depth mask, clearing of the depth buffer is disabled. During rendering of the following views, `glClear` is also re-implemented to have no effect on the color buffer, which stores the views rendered so far.

In contrast to optical multiplexing described in Section 6.1, software multiplexing allows for more flexible and compact display designs. Individual tiles can be displayed by projectors or as tiles of high-resolution LCD displays. By changing the distance between the diffuser and the parallax- barrier, the view zone of the display can be adjusted easily to the requirements of the scene being displayed. Also, the number of tiles, i.e., Chromium servers, does not need to correspond to the number of views. Finally, the elimination of a lenticular sheet for optical multiplexing reduces cross talk and light diffusion and increases image quality.

However, our implementation of software multiplexing incurs, in the best case, an $n$ times network bandwidth overhead for $n$ views, since the application distributes each view separately to the servers. In practice, the overhead is slightly higher because of overlaps at the tile boundaries. We could avoid this overhead by looping over the views on the servers instead of the client node. However, this would require a modified `tilesort` SPU on the client that bucket-sorts geometry based on a *meta view frustum* that spans all the views for each tile. We leave the implementation of such an SPU as future work.

**Alternatives** We have chosen a brute-force approach for distributed multiview rendering with the goal to allow autostereoscopic viewing of existing applications without requiring any changes in their rendering code. However, there are alternatives for multiview rendering that can be more efficient when directly supported by the application. Halle's multiple viewpoint rendering algorithm[21] exploits the geometric structure of epipolar plane images (EPIs) to render scenes from several views simultaneously. While the complexity of this approach is an order of magnitude lower than brute-force multiview rendering, it cannot be applied with existing applications in a non-immersive way. Similarly, the architecture proposed by Stewart et al.[22] relies on a radically new rendering pipeline that is not compatible with existing applications.

## 7. RESULTS

To compare the different display designs, we have built a front-projection display with a lenticular sheet and optical multiplexing similar to the one by Matusik et al.[4] and we have developed a novel prototype with rear-projection, a parallax-barrier and software multiplexing 5. Both systems consist of 16 rendering servers, each one a Windows workstation with an Intel PIV processor running at 3GHz and an NVidia FX5700LE or FX5900 graphics board. They are connected to a client node, also a Windows workstation with Intel PIV processor running at 2.2GHz, through a Gigabit Ethernet network with a single switch.

The physical setup of the novel rear-projection prototype is shown in Figure 7 on the right. The projectors are placed in a $4 \times 4$ grid at a short distance from the display plane, tiling an overall projection area of $72'' \times 48''$. Figure 7, left, shows a close-up of the high resolution image on the projection side of the display, illustrating the multiplexed views using slanted slits. A performance comparison between optical and software multiplexing is

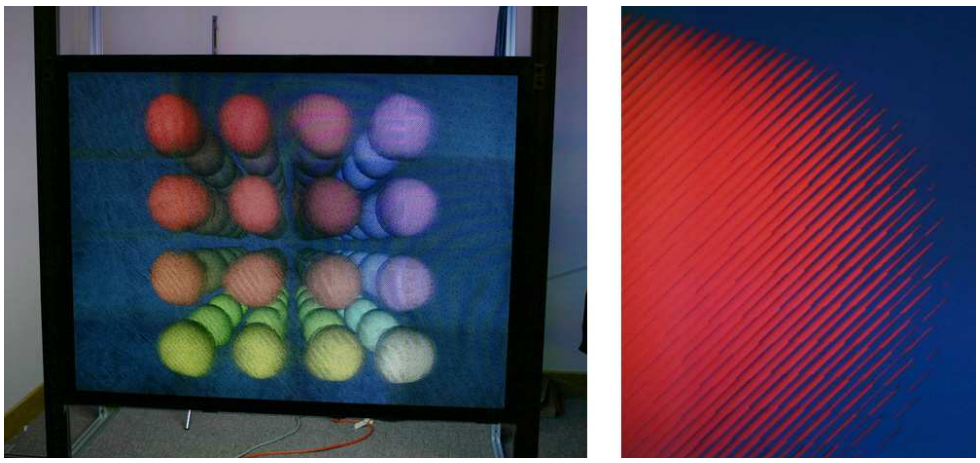


**Figure 7.** Left: Physical setup of the software multiplexing display. Right: Close-up of high resolution image with multiplexed views.

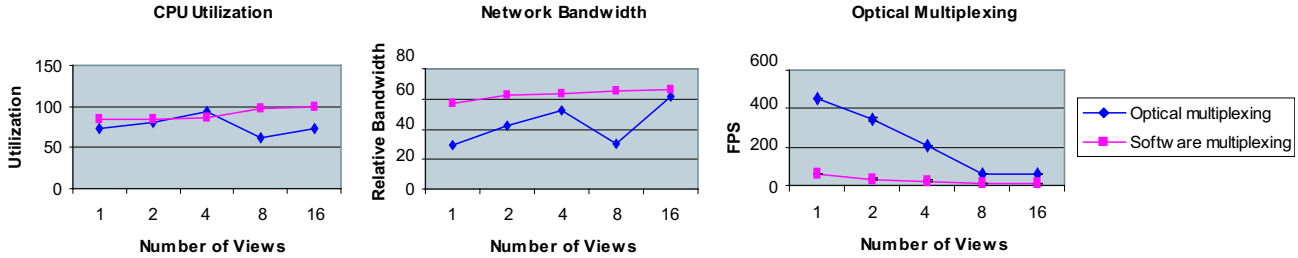



**Figure 8.** Performance comparison between optical and software multiplexing. From left to right: CPU utilization on the application node, network bandwith relative to theoretical peak bandwith (1GBit/sec.), and rendered multiview frames per second.

given in Figure 8. We use a simple test scene with an average of less than one thousand triangles per frame. The major bottlenecks in both cases are either stream processing on the application node CPU, network bandwidth, or rendering performance on the servers. Optical multiplexing seems to be limited by the network for all interesting cases (i.e., more than four views). In our current implementation of software multiplexing, the CPU of the application node traverses and sorts the scene geometry and distributes it over the network for each view. This leads to high loads on the application CPU and on the network already for eight views. As a consequence, the frame rate drops almost linearly with the number of views. We plan to alleviate this problem by implementing a custom tilesort SPU and move multiview rendering completely to the server nodes.

## 8. CONCLUSIONS AND FUTURE WORK

We have investigated the issues in constructing a low-cost parallel rendering system on a network of PCs to drive multiview parallax displays. We presented a classification of rear- and front-projection 3D displays and implementation details of a novel rear-projection parallax-barrier display. We discussed different rendering algorithms for software and optical multiplexing using Chromium. Our results show that optical multiplexing has superior performance but less flexibility in terms of display implementation. This work is a first step towards understanding how to use Chromium to build a parallel rendering system for multiview parallax displays.

We continue to improve the performance of our Chromium SPUs. In particular, implementing a tilesort SPU for multiview frames will reduce the geometry processing bottleneck in the application node and reduce network bandwidth. We are also improving the quality of our multiview parallax-barrier display by investigating solutions to interperspective aliasing.[18]

## REFERENCES


1. J. Konrad and P. Agniel, "Subsampling models and anti-alias filters for 3-d automultiscopic displays," *IEEE Trans. Image Process.* , Jan. 2005. In print.
2. L. Lipton and M. Feldman, "A new stereoscopic display technology: The synthagram," in *Proc. SPIE Stereoscopic Displays and Virtual Reality Systems*, **4660**, pp. 229–235, Jan. 2002.
3. A. Schmidt and A. Grasnick, "Multi-viewpoint autostereoscopic displays from 4d-vision," in *SPIE Stereoscopic Displays and Virtual Reality Systems*, **4660**, pp. 212–221, Jan. 2002.
4. W. Matusik and H. Pfister, "3D TV: A scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes," *ACM Transactions on Graphics (SIGGRAPH 2004)* **23**, pp. 811–821, Aug. 2004.
5. G. Humphreys, M. Houston, Y. Ng, R. Frank, S. Ahern, P. Kirchner, and J. Klosowski, "Chromium: A stream processing framework for interactive graphics on clusters," *ACM Transactions on Graphics (SIGGRAPH 2002)* **21**(3), pp. 693–703, 2002.
6. T. Okoshi, *Three-Dimensional Imaging Techniques*, Academic Press, 1976.
7. B. Javidi and F. Okano, eds., *Three-Dimensional Television, Video, and Display Technologies*, Springer-Verlag, 2002.
8. K. Li, H. Chen, Y. Chen, D. Clark, P. Cook, S. Damianakis, G. Essl, A. Finkelstein, T. Funkhouser, T. Housel, A. Klein, Z. Liu, E. Praun, R. Samanta, B. Shedd, J. P. Singh, G. Tzanetakis, and J. Zheng, "Building and using a scalable display wall system," *IEEE Computer Graphics and Applications* **20**, pp. 29–37, Dec. 2002.
9. R. Raskar, G. Welch, M. Cutts, A. Lake, L. Stesin, and H. Fuchs, "The office of the future: A unified approach to image-based modeling and spatially immersive displays," in *Proceedings of SIGGRAPH 98*, pp. 179–188, (Orlando, FL), 1998.
10. H. E. Ives, "The projection of parallax panoramagrams," *Journal of the Optical Society of America* , pp. 397–409, July 1931.
11. H. Liao, M. Iwahara, N. Hata, I. Sakuma, T. Dohi, T. Koike, Y. Momoi, T. Minakawa, M. Yamasaki, F. Tajima, and H. Takeda, "High-resolution integral videography autostereoscopic display using multi-projector," in *Proceedings of the Ninth International Display Workshop*, pp. 1229–1232, 2002.
12. R. Raskar, M. Brown, R. Yang, W. Chen, G. Welch, H. Towles, B. Seales, and H. Fuchs, "Multi-projector displays using camera-based registration," in *IEEE Visualization*, pp. 161–168, (San Francisco, CA), Oct. 1999.
13. W. Kresse, D. Reiners, and C. Knöpfle, "Color-consistency for digital multi-projector display systems: The heyewall and the digital cave," in *IPT / EGVE 2003: Seventh Immersive Projection Technology Workshop and Ninth Eurographics Workshop on Virtual Environments*, pp. 271–280, May 2003.
14. M. Stone, "Color and brightness appearance issues in tiled displays," *Computer Graphics and Applications* **21**, pp. 58–67, Sept. 2001.
15. A. Majumder, Z. He, H. Towles, and G. Welch, "Achieving color uniformity across multi-projector displays," in *IEEE Visualization 2000*, pp. 117–124, Oct. 2000.
16. A. Majumder and R. Stevens, "Color nonuniformity in projection-based displays: Analysis and solutions," *IEEE Transactions on Visualization and Computer Graphics* **10**, pp. 177–188, Mar. 2004.
17. A. Woods, T. Docherty, and R. Koch, "Image distortions in stereoscopic video systems," in *Stereoscopic Displays and Applications IV*, *Proceedings of the SPIE Volume 1915*, pp. 1–13, (San Jose, CA), Feb. 1993.
18. C. Moller and A. Travis, "Correcting interperspective aliasing in autostereoscopic displays," *IEEE Transactions on Visualization and Computer Graphics* **11**(2), pp. 228–236, 2005.
19. S. Tomov, R. Bennett, M. McGuigan, A. Peskin, G. Smith, and J. Spiletic, "Application of interactive parallel visualization for commodity-based clusters using visualization apis," *Computers & Graphics* **28**, pp. 273–278, Apr. 2004.
20. R. Samanta, J. Zheng, T. Funkhouser, K. Li, and J. P. Singh, "Load balancing for multi-projector rendering systems," in *1999 SIGGRAPH / Eurographics Workshop on Graphics Hardware*, pp. 107–116, Aug. 1999.
21. M. Halle, "Multiple viewpoint rendering," in *Computer Graphics*, *SIGGRAPH 98 Proceedings*, ACM Press.
22. J. Stewart, E. P. Bennett, and L. McMillan, "Pixelview: A view-independent graphics rendering architecture," in *Eurographics Workshop on Graphics Hardware*, 2004.