Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

A Real-Time Distributed Light Field Camera

Jason C. Yang   Matthew Everett   Chris Buehler   Leonard McMillan

Computer Graphics Group
MIT Laboratory for Computer Science

Abstract
We present the design and implementation of a real-time, distributed light field camera. Our system allows multiple viewers to navigate virtual cameras in a dynamically changing light field that is captured in real-time. Our light field camera consists of 64 commodity video cameras that are connected to off-the-shelf computers. We employ a distributed rendering algorithm that allows us to overcome the data bandwidth problems inherent in dynamic light fields. Our algorithm works by selectively transmitting only those portions of the video streams that contribute to the desired virtual views. This technique not only reduces the total bandwidth, but it also allows us to scale the number of cameras in our system without increasing network bandwidth. We demonstrate our system with a number of examples.

1. Introduction

Image-based rendering (IBR) has found widespread use because of the ease with which it produces photorealistic imagery. In particular, light fields and lumigraphs can generate realistic images at high framerates with little or no computer graphics modeling. In the past, light field and lumigraph techniques have been limited to static scenes, in large part because a single camera is often used to capture the images.

More recently, image-based rendering systems have been developed around multi-camera configurations. Such systems have found spectacular success in movies such as "The Matrix," in commercials on TV, and even at the Super Bowl. By using large numbers of still cameras, a snap-shot of a dynamic scene can be captured and manipulated using image-based rendering techniques.

A logical extension is to construct an image-based rendering system with multiple video cameras, thus allowing rendering in an evolving, photorealistic environment. Several such systems have been demonstrated [4, 6]. However, because of the large volume of data, relatively small numbers of widely spaced cameras are typically used. Light field techniques are not directly applicable in these configurations, so these systems generally reconstruct a geometric model of the scene to use for virtual view rendering. This reconstruction process is difficult and time-consuming. Some systems with small numbers of cameras have performed real-time geometry correction, but more often they must do so off-line.

In our work, we propose a scalable architecture for a distributed image-based rendering system based on a large number of densely spaced video cameras. By using such a configuration of cameras, we can avoid the task of dynamic geometry creation and instead directly apply high-performance light field rendering techniques. Our system uses a novel distributed light field rendering algorithm to reduce bandwidth issues and to provide a scalable system. Our work makes the following contributions:

1. A system architecture that allows multiple viewers to independently navigate a dynamic light field.
2. A distributed light field rendering algorithm that allows our system to use bandwidth proportional to the number of viewers instead of proportional to the number of cameras.
3. A complete implementation of a real-time, distributed light field camera consisting of 64 commodity video cameras arranged in a dense grid array.

1.1. Previous Work

1.1.1. Light Fields and Lumigraphs

Levoy and Hanrahan [5] and Gortler et al. [2] first introduced light field techniques: rendering new views from a dense

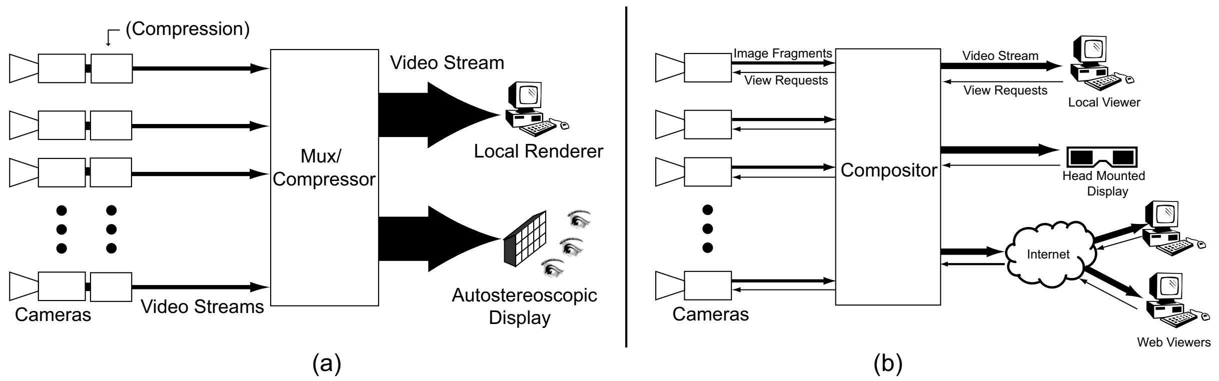

Figure 1: ***Block diagrams for two possible system architectures. (a) The all-viewpoints design, in which the entire light field is compressed and transmitted to a display. (b) The finite-viewpoints design, in which only the portion of the light field that is needed for display is copied from the cameras and transmitted.***

set of reference views. These techniques are noteworthy because they produce photorealistic imagery with little or no geometric model of the scene. This fact not only allows for very fast rendering algorithms but also suggests that a real-time, image-based rendering system is possible, since the time-consuming modeling process is eliminated.

Since the introduction of these papers, there has been a large volume of research on light field techniques. One technique is the dynamically reparameterized light field [3], which allows a user to "focus" the light field on certain portions of a scene. We make use of this technique in our system. We also borrow some ideas from unstructured lumigraph rendering [1], since the cameras in our system are somewhat misaligned due to manufacturing irregularities. Other relevant work includes time critical lumigraph rendering [14], which uses a similar triangulation-based rendering algorithm.

1.1.2. Dynamic Multi-Camera Systems

Separate from light field research, there has also been much interest in building multi-video camera dynamic image-based rendering systems. The Virtualized Reality system of Kanade et al. [4] has over 50 video cameras arranged in a dome shape. This system uses about the same number of cameras as ours, although the dome arrangement is too widely-spaced for pure light field rendering. Instead, a variety of techniques are used to construct dynamic scene geometry [12] for off-line rendering.

At the other end of the spectrum, the image-based visual hull system [6] uses only four cameras to construct a crude geometric scene representation in real-time. However, this system uses silhouette techniques and thus can only represent the foreground objects of a scene.

1.2. Camera Arrays

In recent years, researchers have begun investigating dense camera arrays for image-based rendering. The Lumi-Shelf system [13] uses six cameras in a two by three array. Their use of Region-of-Interests (ROIs) is similar to our selective transmission technique, but they distribute the warping, resampling, and compositing differently than in our system. Most importantly, they use a stereo matching process for geometry correction, which limits the system's performance to 1-2 frames per second. By contrast, we exploit the scalable nature of our rendering algorithm to increase the number of cameras to the point where geometry correction (except for focal plane selection) is no longer needed.

Ooi et al. [10] propose a camera array based on a random access camera, similar to the device we propose later in this paper. The major difference between our approaches lies in the capabilities of the cameras we propose. Their camera allows only pixel addressing, whereas our device incorporates image warping and resampling. They demonstrate a prototype of the camera device, but to our knowledge have not built a system based on it.

More recently, Wilburn et al. [17] have demonstrated a six-camera prototype of a light-field video camera. Their camera is targeted at compression and storage of light fields, and not at real-time interactivity. Naemura et al. [7, 8] present a 16-camera array intended for interactive applications. They compensate for the small number of cameras by using real-time depth estimation hardware. They also employ a simple downsampling scheme (i.e., images are downsampled by 4 in both dimensions for a 4×4 array) to reduce the video bandwidth in the system, which keeps bandwidth low but does not scale well.

2. Design Considerations

A number of considerations affected the design of our light field camera array, including data bandwidth, scalability, and desired uses for the system. We also considered overall system cost when designing our system.

2.1. Data Bandwidth

Light fields are well-known for requiring large memory resources. In moving to a dynamic, video-based system, the data management problems become magnified. Thus, one of the critical design criteria is to keep total data bandwidth (i.e., the amount of data transferred through the system at any given time) to a minimum.

2.2. Scalability

The image quality of light field rendering improves with increasing number of images in the light field. Thus, the system should be able to accommodate a large number of video cameras for acceptable quality. Ideally, we would like to be able to increase the number of video cameras without greatly increasing the data bandwidth of the system.

2.3. Desired Uses

A dynamic light field system has many potential uses, some of which may be better suited to one design than another. For example, a light field camera array may be used to drive a high-resolution monitor with special lenses that allow autostereoscopic viewing (i.e., a true 3D television). Static versions of such devices have been demonstrated [3]. In this application, the camera must have the bandwidth to deliver the full light field to the display. On the other hand, to generate real-time stereo pairs for a head-mount display, the light field camera only needs the bandwidth to generate two virtual views at a time.

3. System Design

In designing our system, we evaluated two possible system configurations. The fundamental difference between these two designs is in the number of output views they can deliver to an end user. The first system is a straightforward design that essentially delivers all possible output views (i.e., the whole light field) to the end user at every time instant. We call this system the “all-viewpoints” system.

The second system, the one we chose to implement, is constrained to deliver only a small set of requested virtual views to the end user at any time instant. We call this system the “finite-viewpoints” system.

The all-viewpoints system predictably has high bandwidth requirements and poor scalability. However, it offers the most flexibility in potential uses. The finite-viewpoints system is designed to be scalable and utilize low data bandwidth. On the other hand, it limits the potential uses of the system.

First, we briefly review the all-viewpoints design to illustrate the various tradeoffs involved.

3.1. All-Viewpoints Design

The all-viewpoints design is the simpler of the two that we considered (see Figure 1). In this design the camera array delivers all of the video streams to a display device. We call this the “all-viewpoints” design because any virtual viewpoint can then be synthesized at the display device.

The advantage of this approach is that *all* of the light field data is available at the display device. This feature allows, for example, the light field to be recorded for later viewing. In addition, this design could permit the light field to generate any number of virtual views *simultaneously*, given a suitable multi-user autostereoscopic display device.

The primary disadvantage is that the data bandwidth required to transmit large numbers of video streams is large. In addition, the bandwidth increases with the number of cameras, making the design difficult to scale.

Considering that a single uncompressed CCIR-601 NTSC video stream is 26.2 Mbytes/sec, a system with 64 cameras would require aggressive compression capabilities. An optimal compression device would encode all video streams simultaneously, which could potentially lead to an asymptotic bound on the total output bandwidth as the number of cameras are increased. However, the input bandwidth to any such compression device would increase unbounded. The most likely compromise is to place compression capabilities on the cameras themselves, which would reduce the bandwidth by a constant factor but not bound it.

3.2. Finite-Viewpoints Design

We decided not to pursue an all-viewpoints design because of the bandwidth requirements. Instead, we decided to implement a “finite-viewpoints” design, which trades off viewing flexibility for a great reduction in bandwidth.

In the finite viewpoint design (see Figure 1b), cameras are treated as “random access” video buffers whose contents are constantly changing. The cameras are individually addressable, and subsets of pixels can be copied from their video buffers. The key idea is that the individual contributions from each camera to the final output image can be determined independently. These “image fragments” can then be combined together later (e.g., at a compositing device or display) to arrive at the final image.

The system shown in Figure 1b operates as follows. An end user at a display manipulates a virtual view of the light field. This action sends a virtual view request to a central

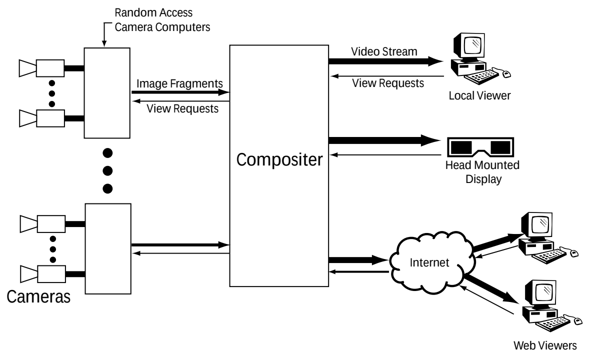

Figure 2: ***A block diagram of our system implementation. Random access cameras are simulated by normal cameras connected to a bank of computers. These computers send image fragments to the image compositor, which assembles the final image. Users view images from the compositor on their displays.***

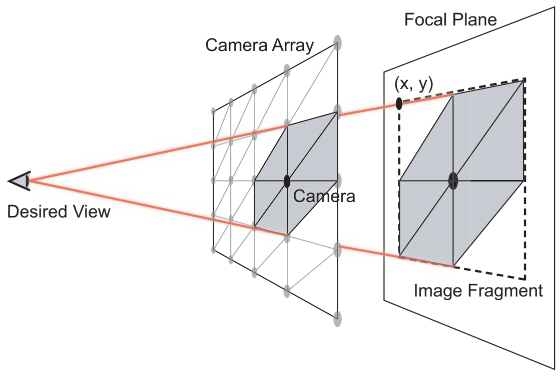

Figure 3: ***A diagram of the rendering algorithm. All triangles associated with a camera are projected onto the focal plane. These triangles are rendered from the desired viewpoint with projective texture mapping. The result is an image fragment that is transmitted for later compositing.***

compositing engine. The compositor then sends the virtual view information to each of the cameras in the array, which reply with image fragments representing parts of the output image. The compositor combines the image fragments and sends the final image back to the user's display.

The system transfers only the data that is necessary to render a particular virtual view. In this way, the total bandwidth of the system is bounded to within a constant factor of the size of the *output image*, because only those pixels that contribute to the output image are transferred from the cameras. In general, the system may be designed to accommodate more than one virtual viewpoint, as long as the number of viewpoints is reasonably small. Note that there is some bandwidth associated with sending messages to the cameras. However, with the appropriate camera design, these messages can in fact be broadcast as a single message to all the cameras simultaneously.

Along with a reduction in bandwidth, this system design has other advantages. First, it more readily scales with additional cameras, as the total bandwidth is related to the number and size of the output streams instead of the number of input streams. Second, the display devices can be extremely lightweight, since only a single video stream is received. A light field camera employing this design could conceivably be manipulated over the internet in a web browser.

The main disadvantage of the finite viewpoint system is that the entire light field is never completely transferred from the cameras. Thus, it is difficult to record the light field for future playback, and it is impossible to display all of the data at once (e.g., on a stereoscopic display).

4. Implementation

We have implemented a light field camera array based on the finite-viewpoints design. In the interest of keeping system costs down, we constructed the system from commodity parts. First, we describe the basic light field rendering algorithm that our system implements. Then we describe the two key system components, the random access cameras and the compositing engine, and how they interact (see Figure 2). Finally, we discuss other details that are necessary for a complete system.

4.1. Rendering Algorithm

The actual implementation of our system is intimately tied to our choice of light field rendering algorithm. The algorithm that we use is related to the hardware-assisted algorithms described in [1, 2, 3, 14]. These algorithms are well-suited to our system since they construct the output image on a camera-by-camera basis.

First, we assume that we know the positions and orientations of all of our cameras. While we do not require that the camera positions strictly conform to a regular grid, we do assume that the cameras have a fixed regular topology. For example, in our system, we have configured 64 video cameras in an eight by eight square grid. We triangulate this grid to arrive at a fixed topology for rendering (see Figure 3).

As in [3], we render the light field with respect to a user-specified focal plane. Conceptually, rendering proceeds as follows. For each camera (i.e., vertex) in the grid, the triangles that are connected to the vertex are projected through the desired view onto the focal plane. These triangles are then rendered from the desired point of view, with projective texture mapping of the camera's current image. The alpha values of the triangles are set such that the camera's vertex has alpha one and all other vertices have alpha zero. These rendered triangles constitute the *image fragment* that this particular camera contributes to the final image. We accumulate all such image fragments to arrive at the final image.

The details of how we distribute this algorithm depends on the operation of the random access cameras, which we discuss next.

4.2. Random Access Cameras

A random access camera returns a portion of its current image in response to a request for data. The amount of "intelligence" on the camera determines how the light field algorithm is distributed.

A simple random access camera knows nothing about itself or its neighbors, and looks like a frame buffer to the compositor. In this case, the compositor knows all information about the system, including camera positions, focal plane position, etc. During rendering, the compositor determines which pixels are needed from each camera and directly reads them out of the cameras' frame buffers and performs all texture filtering, alpha blending, and compositing. This is the type of camera module described in [10].

However, we have found that a random access camera for light field applications can benefit greatly from image processing capabilities. For example, as discussed previously, rendering the light field involves projective texture mapping, which is a simple perspective transformation applied to a spatially coherent part of the camera's image buffer. Thus, an intelligent random access camera should be able to warp its image fragment given a desired triangular output region and a 2D perspective transformation matrix. The compositor could then generate requests for triangular patches instead of individual pixels and perform only alpha compositing.

A random access camera can be further improved by having knowledge of its position in the world. In this case, the compositor could send just a desired 3D triangle along with the virtual view information and focal plane information. The required perspective image warp can then be derived from that information.

In our system, we assume the camera has knowledge of its own position and the positions of its neighboring cameras. Given this information, the camera itself can determine its contribution to the final output image given just the virtual view information and the focal plane. All cameras need the same information, so it can be broadcast to all the cameras at one time.

We decided to use this camera model because of the simplified communication protocol. However, it is not clear that this choice is always best. For example, cameras that have no knowledge of their neighbors might be useful for a fault tolerant system. A sophisticated compositor could reconfigure the camera topology on-the-fly in the case of camera failures.

4.2.1. Simulating Random Access Cameras

Unfortunately, the type of random access camera that we envision does not yet exist. With the advent of cheap CMOS imagers, however, these cameras could certainly be built on a single chip.

In the meantime, we can simulate such cameras with a regular video camera connected to a graphics hardware-equipped computer. The video frames are downloaded into the computer's memory. As view requests are received over the network, the computer warps a small part of the video frame and sends the resulting image fragment back to the compositor. The image fragment is a small rectangular patch with associated 2D coordinates (see Figure 3). These coordinates tell the compositor where to position the image fragment in the final image. The fragments are also tagged, so the compositor can determine to which virtual view and focal plane they belong. Fragments can also be compressed, resulting in even further bandwidth reduction. Our system uses simple JPEG compression for this purpose.

We have found that standard PCs are sufficient for simulating a random access camera. In fact, a single computer can easily simulate more than one camera. In our system, we use a single computer to simulate up to 16 random access cameras. The computer stores the position and neighbor information for each camera and performs image processing and compression.

4.3. Image Compositor

The image compositor is the central communication point in our system. End users connect to the compositor when they want to view the light field data.

The user's display program sends a view request to the compositor whenever the display needs an update. The view request consists of a virtual view and focal plane. The virtual view is specified by position, orientation, and field-of-view, and the focal plane is specified by a plane equation.

The compositor broadcasts the view request information to the random access cameras. It then clears a new frame buffer and waits for the appropriate image fragments to return from the cameras. When a fragment arrives, the compositor aligns the upper-left corner of the fragment with the proper 2D coordinates returned by the camera. It then adds the fragment into the frame buffer.

When all fragments have been received, the completed output image is returned to the user's display. Because of the small amount of data involved, the entire process happens very quickly and there is very little latency. Also, the system maintains nearly constant throughput because light field rendering is insensitive to the virtual view selection or scene complexity.

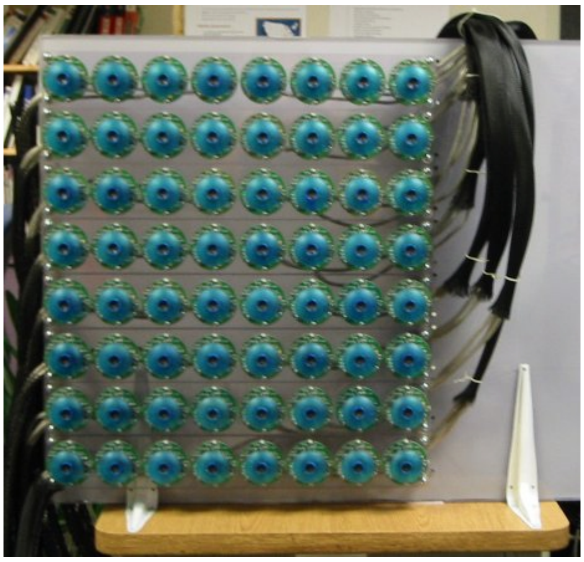

Figure 4: ***A photo of our 64-camera light field camera array. The cameras are arranged in rows of eight.***

4.4. System Details

4.4.1. Hardware

Cost was a significant constraint in choosing the hardware for our system. The camera array itself consists of 64 OrangeMicro iBot firewire video cameras (see Figure 4). These cameras, while cheap, have a few major disadvantages. First, they can not be genlocked, which means that the images in our light field are not exactly synchronized in time. We have developed a way to synchronize to within one frame, which is sufficient for a prototype system (see below). Second, the color controls on the cameras do not behave in a consistent manner, which makes color calibration difficult (see below). Third, the cameras have noticeable radial distortion, which complicates camera calibration and introduces geometric distortions in renderings (see below).

To build the array, we stripped the cameras of their casings and rigidly mounted them to a Plexiglas board. The cameras are organized into groups of eight, which are connected to two firewire hubs. Rows of eight cameras are then connected to PCs, which act as the random access camera simulators for those cameras. Currently, the cameras are arranged in an eight by eight grid; however, we can reconfigure the array by rearranging the eight-camera modules.

Currently, we use six different computers as random access camera simulators. They range from 1.5GHz to dual 1.7GHz Pentium 4 PCs. Each computer has two firewire channels for grabbing video from the cameras. The two fastest computers each simulate 16 random access cameras, while the other four computers each simulate 8 cameras. The computers are equipped with nVidia Quadro2 Pro graphics cards, and they are attached via a dedicated network to the compositing machine.

The compositor computer is a slower 1.5GHz Pentium 4 computer, and it does not require an advanced graphics card. The compositor drives a display that users can use to view the light field. In the future, users will be able to connect remotely (e.g., over the web) to the compositor machine, but we have not enabled this capability at this time.

4.4.2. Camera Calibration

For quality light field rendering, it is important to know accurately the positions, the orientations, and the internal parameters (e.g., focal length, etc.) of the cameras. Calibrating a camera is a tedious process; individually calibrating 64 (or more) cameras is intractable. To make the process easier, we have developed a largely automatic process.

First, we manually calibrate the intrinsics of one of the cameras. The intrinsics include focal length, principal point, and two radial distortion parameters. We do this using Zhang's calibration software [18]. Initially, we assume that all of the cameras have the same intrinsics as this first camera. We relax this assumption later in the calibration process.

Next, we darken the room and move a point light source in front of the camera array. Each camera tracks the position of this light source, which is easy to see in the dark environment. We acquire about 100 points of data.

We then run standard structure-from-motion computer vision algorithms [15, 16] on this point data to compute an initial estimate of the cameras' positions and orientations as well as the 3D positions of the point light sources. In this step, we have assumed identical intrinsics, which causes the structure-from-motion process to yield noisy results.

Finally, we relax the identical intrinsics assumption and refine the initial estimate using a large nonlinear optimization. This optimization improves the position, orientation, and intrinsics of each camera. (The 3D point light source positions are improved as well, but we do not use this data.) Using this process, we generally achieve a calibration with reprojection errors of less than 1 pixel, which is suitable for image-based rendering applications.

4.4.3. Color Calibration

When mixing images from many different cameras, it is important to match the colors between cameras. Again, independently adjusting the contrast, brightness, and white balance controls for 64 cameras is a tedious process. To assist in this task, we have implemented the simple automatic process described in [9].

Unfortunately, our low-cost cameras have defective color controls (similar to the problem reported in [9]), so the automatic process does not always succeed on all cameras. In these cases, we must adjust some color controls manually.

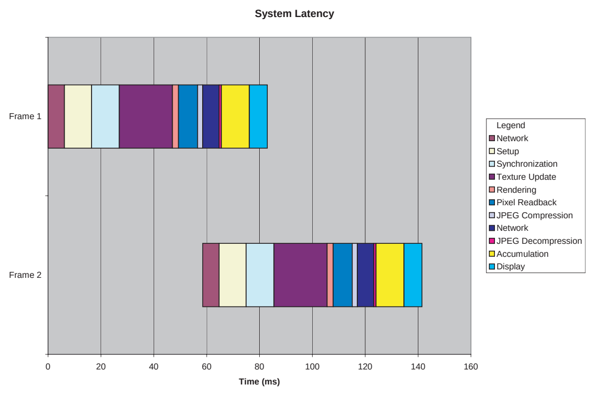

Figure 5: ***The round trip latency of our system, broken down into its components. Grabbing images from the cameras happens asynchronously and occurs in the background. Therefore, the time involved in this operation does not appear in our latency graph.***

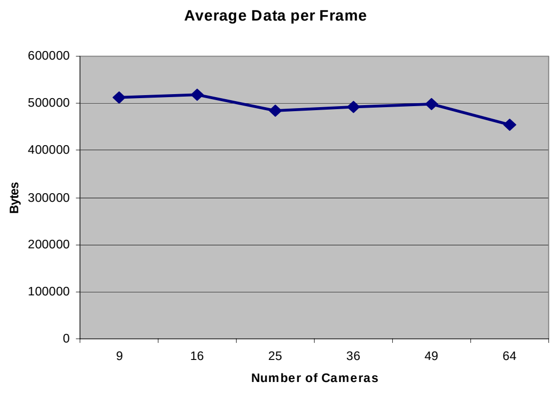

Figure 6: ***The average data per frame that is transmitted from the camera computers to the compositor. Bandwidth stays roughly constant for different numbers of cameras. These values reflect uncompressed data.***

4.4.4. Synchronization

Although our cameras are not genlocked, we can ensure that all the images within our light field are not off by more than one frame. When the system first starts up, all the computers synchronize their internal clocks. Once they finish this step, their clocks differ by no more than 5-10ms.

Then, the camera computers agree on a schedule for when they will download images from the cameras. Since their clocks are aligned, they will all receive an image from their respective cameras within 5-10ms of each other. Because the cameras run at only 15fps, this scheme is sufficient to keep the light field images to within a frame of one another.

5. Results and Discussion

Our system generates images at an average of 18 frames per second, with 320 by 240 video coming from the cameras. We can achieve this rendering rate because rendering and capture happen asynchronously on our system. Unfortunately, the frame rate of the system is limited by the slowest camera computers. Our fastest computers easily run at 30 frames per second, so we know the system can attain faster speeds with minor computer upgrades.

The latency is only 80 milliseconds, which makes the system extremely responsive, but it is probably still inadequate for head-tracked display [11]. Figure 5 breaks down the latency of the system into its various components. We can see that most time is spent copying texture data to and from the graphics cards. This is an unfortunate artifact of simulating random access cameras; the real cameras would not suffer this penalty. The synchronization delay represents our scheme for attempting to synchronize our ungenlocked cameras. We can also see that much time is spent compositing and displaying the final images. However, this time is essentially free, as we can hide it with pipelining as shown by the second row of the graph. The rendering and image compression tasks take the least amount of time, with network communication not too far behind. The “setup” category covers timings that did not belong in other categories.

We also conducted experiments to verify the scalability of the system. We ran the system with varying numbers of cameras and measured the data bandwidth of the image fragments entering the compositor. We took care to always use cameras spread over the whole array so that we could render the same size output image. For example, when testing four cameras, we used the cameras at the corners of our array.

We used a pre-programmed camera path and averaged the results over 20 seconds of running. Ideally, this measure should show that the bandwidth of our system does not increase as more cameras are added. See Figure 6. Note that the bandwidth is not identical in all cases because we always transfer rectangular image fragments, which might contain extraneous pixels that are known to be zero. We actually see a reduction in bandwidth as we increase the number of cameras, which may be caused by fewer extraneous pixel transfers at finer camera granularities.

We have used our light field camera to view and manipulate a wide range of data. In Figures 7, 8, and 9 we demonstrate the variable focus capability of our system. These figures also reveal one of the system’s rendering artifacts: double images. These types of artifacts are characteristic of light field rendering, especially in light fields with only 64 images. We can improve image quality by increasing the number of cameras and redesigning the spacing between them. For example, by using Point Grey’s Firefly cameras in the extended CCD form factor, we can pack the cameras much more tightly than our current design. Additionally, the double images could be replaced with perhaps less objectionable blurring artifacts by blending together more than 3 images per pixel. This type of blending could be implemented by using a finer triangle grid with multiple blending weights at each vertex, as is done in [1].

Figure 12 shows the 64 raw images taken in an instant of time. The person is these images is jumping in mid-air while the camera records him. Figure 11 shows a cross-fusion stereo pair that we synthesized with our system. In our system, our frame rate is inversely proportional to the number of virtual views at any one time. In the case of the stereo pair, the frame rate for each view would be cut in half. For more examples of the system in action, please see the accompanying video.

6. Conclusion and Future Work

We have presented the design and implementation of a real-time distributed light field camera. Because of our finite-viewpoints design model, we are able to reduce the necessary bandwidth of the system while still allowing high performance. Our design is also very scalable, which permits the use of large numbers of cameras. Using a large number of low-cost cameras permits us to use no geometry correction, a bottleneck of many previous dynamic image-based rendering systems.

We have demonstrated the feasibility of this approach by implementing the system with readily available, low-cost components. We have described techniques for calibrating both the pose and the color responses of the individual cameras, tasks that are crucial for success. The 64-camera system renders at 18 frames per second with 80 ms latency. We believe that with further tuning, such systems can run even more efficiently.

In the future, we would like to expand the size of our array to 128 or 256 cameras. Such a large array becomes more feasible as the prices of consumer video cameras continue to drop. We would also like to experiment with other camera configurations (e.g., cylindrical or linear arrays) and more sophisticated rendering algorithms.

We are also continuing to investigate "smart" camera technology. With the advent of CMOS imaging, these types of cameras will eventually become prevalent. In this paper, we have assumed a relatively unambitious camera with the ability to warp and resample selected portions of its frame buffer. Cameras with even more intelligence might form the basis of better systems, such as self-organizing camera webs or distributed tracking systems.

7. Acknowledgements

We would like to thank Nathan Ackerman and Manu Seth for their help on this project as well as our anonymous reviewers

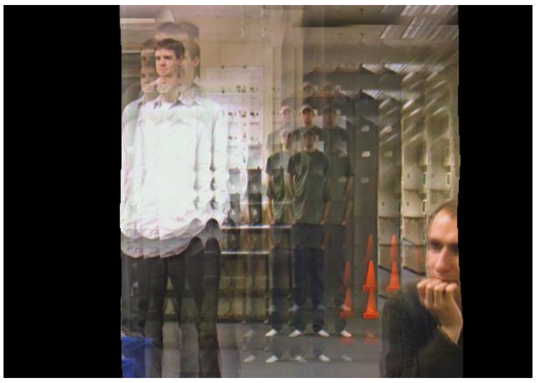

Figure 7: ***Focusing on the person to the right.***

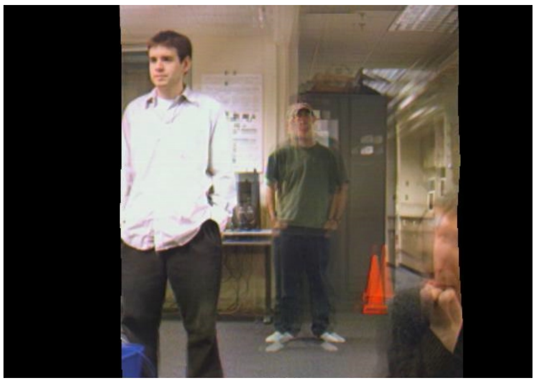

Figure 8: ***Focusing on the person to the left.***

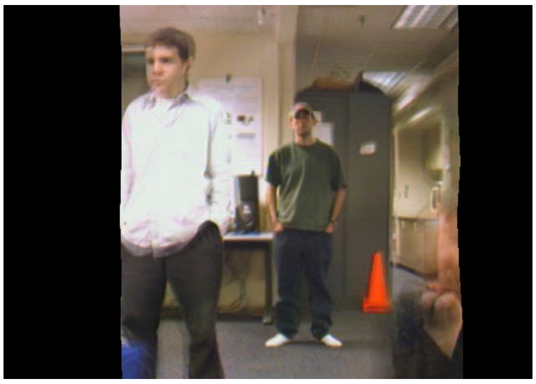

Figure 9: ***Focusing on the person in the center.***

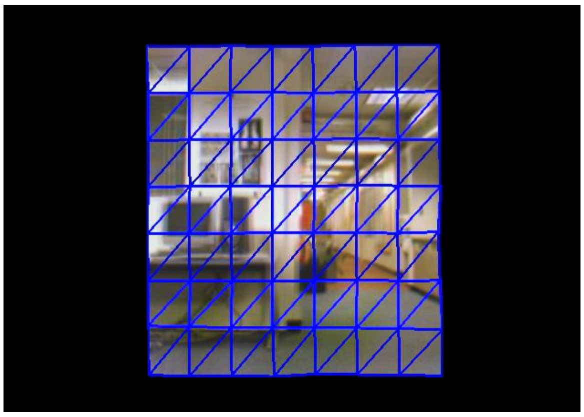

Figure 10: ***Camera positions are triangulated during rendering. This view was taken behind the camera array and shows the contributions from all 64 cameras.***

for their invaluable comments. This work was supported by grants from Nippon Telegraph and Telephone, Inc. (NTT), NSF (CCR-9875859), and MIT Project Oxygen.

References

1. Chris Buehler, Michael Bosse, Leonard McMillan, Steven J. Gortler, and Michael F. Cohen. Unstructured lumigraph rendering. In Eugene Fiume, editor, *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 425–432. ACM, ACM Press / ACM SIGGRAPH, 2001. 2, 4, 8

2. Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *SIGGRAPH 96*, pages 43–54, 1996. 1, 4

3. A Isaksen, L. McMillan, and S. Gortler. Dynamically reparameterized light fields. *SIGGRAPH '00*, pages 297–306, 2000. 2, 3, 4

4. Takeo Kanade, Peter Rander, and P.J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia, Immersive Telepresence*, 4(1):34–47, January 1997. 1, 2

5. M. Levoy and P. Hanrahan. Light field rendering. *SIGGRAPH 96*, pages 31–42, 1996. 1

6. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven Gortler, and Leonard McMillan. Image-based visual hulls. In Kurt Akeley, editor, *Proceedings of SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 369–374. ACM, ACM Press / ACM SIGGRAPH, 2000. 1, 2

7. Takeshi Naemura and Hiroshi Harashima. Real-time video-based rendering for augmented spatial communication. *Visual Communication and Image Processing 1999 (SPIE)*, pages 620–631, 1999. 2

8. Takeshi Naemura, Junji Tago, and Hiroshi Harashima. Real-time video-based modeling and rendering of 3d scenes. *IEEE Computer Graphics and Applications*, pages 66–73, 2002. 2

9. Harsh Nanda and Ross Cutler. Practical calibrations for a real-time digital omnidirectional camera. *CVPR 2001 Technical Sketch*, 2001. 6

10. Ryutaro Ooi, Takayuki Hamamoto, Takeshi Naemura, and Kiyoharu Aizawa. Pixel independent random access image sensor for real time image-based rendering. *IEEE Intern. Conf. Image Process. (ICIP2001)*, 20(3):193–196, 2001. 2, 5

11. Matthew J. P. Regan, Gavin S. P. Miller, Steven M. Rubin, and Chris Kogelnik. A real-time low-latency hardware light-field renderer. *SIGGRAPH '99*, pages 287–290, 1999. 7

12. Hideo Saito, Shigeyuki Baba, Makoto Kimura, Sundar Vedula, and Takeo Kanade. Appearance-based virtual view generation of temporally-varying events from multi-camera images in the 3d room. *Second International Conference on 3D Digital Imaging and Modeling*, 1999. 2

13. Hartmut Schirmacher, Li Ming, and Hans-Peter Seidel. On-the-fly processing of generalized lumigraphs. *Eurographics 2001*, 20(3), 2001. 2

14. Peter-Pike Sloan, Michael F. Cohen, and Steven J. Gortler. Time critical lumigraph rendering. *1997 Symposium on Interactive 3D Graphics*, pages 17–24, 1997. 2, 4

15. Richard Szeliski and Sing Bing Kang. Recovering 3d shape and motion from image streams using nonlinear least squares. *JVCIP*, 5(1):10–28, 1994. 6

16. Bill Triggs, Philip McLauchlan, Richard Hartley, and Andrew Fitzgibbon. Bundle adjustment–a modern synthesis. *Vision Algorithms: Theory and Practice*, pages 298–373, 1999. 6

17. Bennett Wilburn, Michal Smulski, Kevin Lee, and Mark A. Horowitz. The light field video camera. *Proceedings of Media Processors 2002, SPIE Electronic Imaging 2002*, 2002. 2

18. Z. Zhang. A flexible new technique for camera calibration. *Microsoft Research Technical Report: MSR-TR-98-71*, 1998. 6

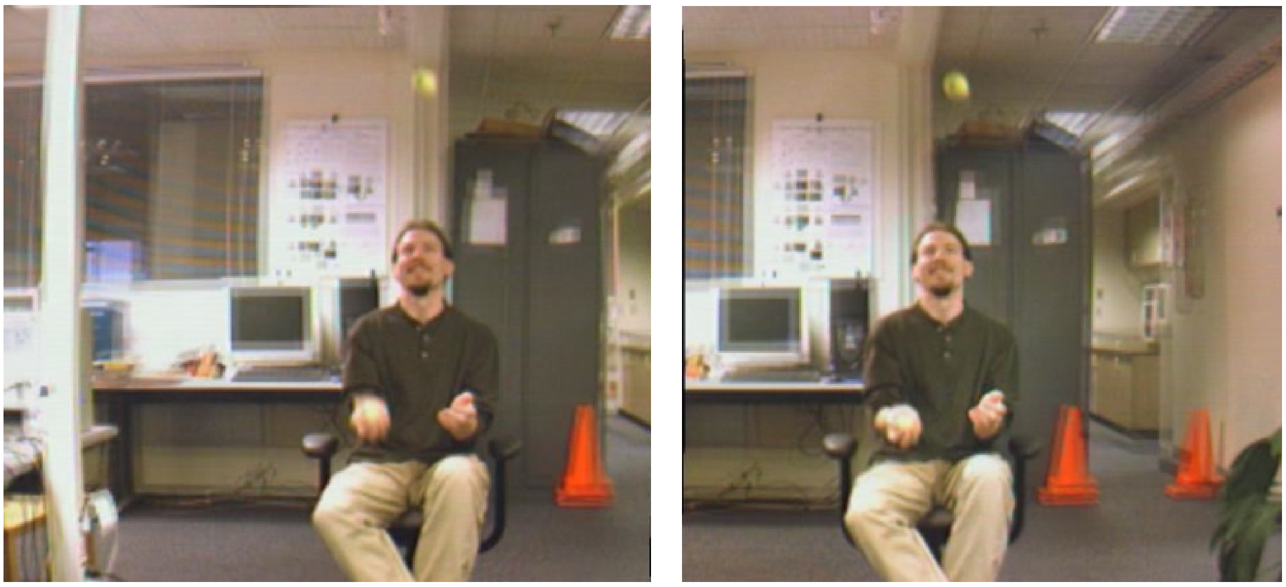

Figure 11: *Cross-eyed stereo pair synthesized from our light field camera.*

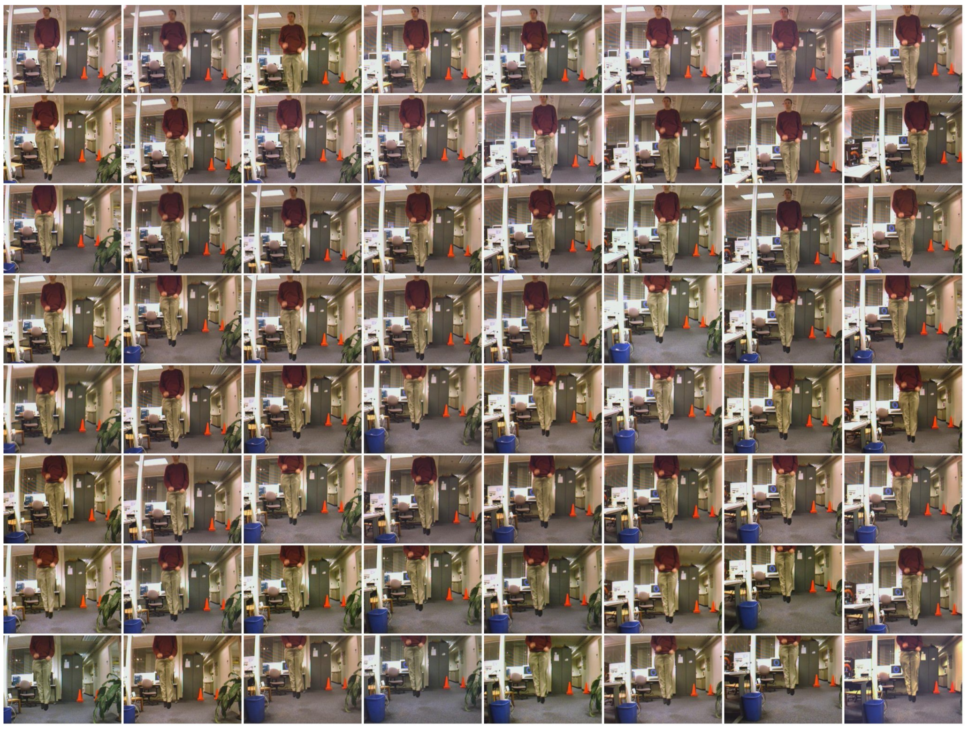

Figure 12: *An array of raw images taken with our light field camera. The person in this light field is jumping in mid-air. Note that these images are unrectified, so the variations in the cameras' viewing directions are readily apparent.*