A Hardware Ray-Space Data Renderer for Interactive Augmented Virtuality

Yukio Sakagawa, Akihiro Katayama, Daisuke Kotake, and Hideyuki Tamura
Mixed Reality Systems Laboratory Inc.
E-mail: {sakagawa, katayama, kotake, tamura}@mr-system.co.jp

Abstract

This paper describes the design and implementation of a dedicated hardware architecture for consumer PCs to interactively render photo-realistic images using Ray-Space representation (an image-based rendering technique). In Ray-Space representation, a Ray-Space map is used to describe the light rays in 3D space. The Ray-Space map and captured image data of an object are stored in this architecture. The reconstructed images are transferred to a 3D graphics board where they are composited in a scene with geometric model data. Rendering with the current prototype is fast enough to be used in an application like an interactive virtual shopping mall where the scenes are generated from both 3D geometric data and Ray-Space data.

Keywords: augmented virtuality, graphics hardware, image-based rendering, computer vision and computer graphics for MR, applications and systems of MR.

1 Introduction

Embracing the concept of Mixed Reality [10], we are developing researches in Augmented Virtuality. Our approach is to enhance or augment virtual environments with data from the real world.

Image-based rendering (IBR) techniques such as HoloMedia [7], Light Field [8], Lumigraph [6], the Delta Tree [5] and Ray-Space method [9] use images of real objects to recreate photo-realistic images of complex shaped objects without any explicit geometric model.

Rendering realistic scenes with geometric models requires very detailed geometric information. With IBR techniques, the rendering cost is independent of the geometric complexity of the scene. However, with the advances in technology of 3D graphics hardware for rendering geometric primitives, most of the processing load has been transferred from the host CPU to dedicated hardware. IBR techniques could also benefit from dedicated hardware, in particular to handle the large data sets required by these techniques.

This paper describes a rendering architecture specific for Ray-Space data. This dedicated architecture alleviates the processing load and memory usage of the host system.

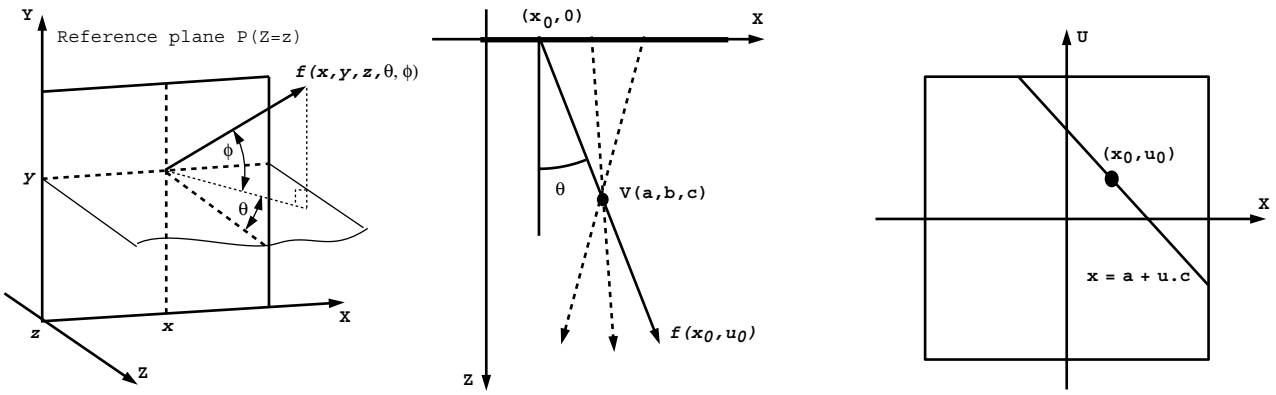
A few hardware architectures have been developed to render images in real time from non-polygonal data, including volumetric and IBR data.

The VolumePro [12] is an architecture dedicated to ray-casting on volume data. It relies on a graphics card to display the results. The Talisman architecture [15] performs 2D warping of rendered image layers to increase rendering performance of polygonal-based data. The WarpEngine [13] uses images with per-pixel depth to perform 3D warping with interpolation. The work of Regan et al. on Low-Latency Light-Field Renderer [14] is the closest to our work, since we consider similar Plenoptic functions [1]. Our architecture has the same advantage over conventional 3D graphics pipeline: the geometric transformations and rasterization can be done at once. However, unlike Regan et al. [14], our main concern is not low-latency but the usability of the renderer for practical interactive applications.

We developed an interactive augmented virtuality system: the CyberMirage [3, 4, 16]. This system displays photo-realistic virtual environments composed of Ray-Space data and geometric model data. It uses our dedicated Ray-Space renderer hardware implementation to achieve interactive frame rates.

Our implementation demonstrates that the processing of Ray-Space data can be transferred efficiently from the host CPU to a hardware architecture. Our current prototype is a first step towards a practical IBR architecture usable in real world applications.

The rest of the paper describes our architecture in details. In Section 2 we describe the Ray-Space data representation. The CyberMirage system is explained in Section 3. Section 4 shows the design of the hardware Ray-Space renderer architecture, and its implementation is described in Section 5. Section 6 describes the layers of driver interface between the application software and the hardware. Finally, Section 7 evaluates the performance of the prototype



(a) Real space (b) Ray space

Figure 1 Relationship between real space and ray space

implementation, and Section 8 describes future work and gives a conclusion.

2 Ray-Space Representation

The Ray-Space representation, also known as Ray Based representation, is an image-based representation. It is a simplified version of Adelson's plenoptic function [1], that has fewer dimensions.

The Ray-Space and Light Field representations are similar in their principles, but they are different in their implementations.

Description of the Rays in 3D Space

In Ray-Space representation, a ray of light passing through a reference plane $P(Z = z)$ is defined by the position (x, y) where the ray intersects the plane and the direction (θ, ϕ) of the ray (see Figure 1). If we ignore the vertical parallax, the degrees of freedom of the ray can be restricted to (x, θ). Let $u = \tan\theta$, then the set of rays passing through a viewpoint $V = (a, b, c)$ in real space is mapped to the straight line

$$x = a + u \cdot c \tag{1}$$

in (x, u) space as Figure 1(b) shows.

An image captured by a camera can be considered as the set of rays that pass through the lens focal point. In this way, the (x, u) space can be filled if several images are collected. If the captured images were not enough to fill out all the (x, u) space, then it is necessary to use an interpolation method. The interpolation method can be as simple as nearest neighbour, or more complex such as algorithms based on the shape of the object. The information about the (x, u) space is stored into a map that we call *Ray-Space map* or *RS map*. Conversely, the image from a new point of view can be generated by reading the data from the (x, u) space on the straight line, in (x, u) space, that corresponds to the new point of view. Nevertheless, when only one reference plane is used, the direction of the rays that can be described is limited. To solve this problem, multiple reference planes arranged radially about the Y axis can be used to describe rays in any direction.

Since the adopted Ray-Space representation ignores vertical parallax, images are generated with vertical distortion. This results from the discrepancy in the heights of a point's projections on imaging planes at different distances from the point, as shown in Figure 2. In this figure, consider a point A on the Y axis. Let the distance from P to the Y axis be l_P, and the distance from Q to Y axis be l_Q. Let the height of point A on the imaging plane of P be h_P, and that of Q be h_Q. The focal distance is set to unity. Then,

$$h_Q = \frac{l_P}{l_Q} h_P \tag{2}$$

To compensate this distortion, we assume the object is located at the reference plane and we apply the correction along the Y axis using equation 2. This assumption works best for objects that have little variation in depth in the vertical direction.

3 The CyberMirage

The CyberMirage is an interactive augmented virtuality system [3, 4, 16]. This system displays photo-realistic virtual environments from Ray-Space data and geometric model data.

In CyberMirage, we use a VRML scene tree that contains nodes describing both geometric model data and Ray-Space data. However, Ray-Space data do not provide the 3D shape data explicitly and hidden surfaces elimination is not straightforward. The depth of these objects is approximated by the depth of their reference plane. Using this approximation,

mutual occlusion can be computed for all the objects in the scene tree.

Figure 3 shows a simple example of such representation. Consider the room, table and coach of Figure 3(a). From the world coordinate C_0, the coordinate transformation T_2 is used to obtain C_2, the coordinate system of the room. The desk and the coach in that room are described in the coordinate systems C_3 and C_4, which are derived by the coordinate transformations T_3 and T_4, respectively, from the coordinate system C_2. Figure 3(b) shows the tree structure that represents these relations. To render this scene, a tree like the one of Figure 3 is traced starting from the left node, through all nodes by a depth-first search and the scene is drawn based on the information written in the nodes. In this process, the image of the geometric shape model data is generated by ordinary rendering process. In order to render from the Ray-Space data, on the other hand, it is necessary to have the position of the viewpoint and the direction of view line as seen from the reference plane, and the field of view. The view position and direction can be determined by successively applying the coordinate transformations that appear while traversing the tree. In the case of Figure 3, let the position of the camera in coordinate system C_1 be P_1, then the position of the camera P_5 on the Ray-Space data coordinate C_5 is given by

$$P_5 = T_5^{-1} \cdot T_3^{-1} \cdot T_2^{-1} \cdot T_1 \cdot P_1 \qquad (3)$$

The view direction of the camera is obtained in a similar fashion.

4 Hardware Architecture

The present architecture is intended for applications where several photo-realistic images generated from Ray-Space data are composited in a scene rendered from conventional 3D geometric primitives.

To achieve this goal, the architecture needs to generate images in true color with α values (translucency), have enough memory to hold the Ray-Space information of several objects, and be fast when rendering the images. Due to memory limitations, Ray-Space data without vertical parallax was adopted.

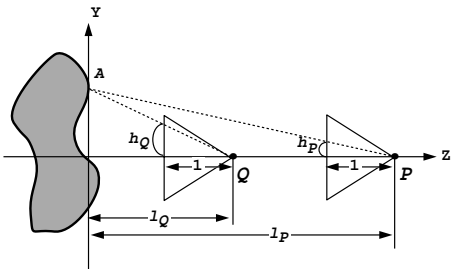
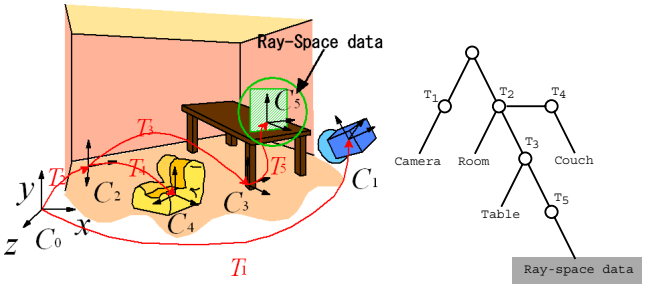


Figure 2 Compensation of vertical distortion.



(a) Virtual space (b) Tree structure

Figure 3 Virtual space representation

An off-the-shelf 3D graphics hardware renderer board is used for the rendering of the geometric data of the scene, and also to display the images generated by the Ray-Space renderer. Figure 4 shows the architecture of the system.

The Ray-Space Accelerator board holds and manages all the information and functions necessary to generate images from Ray-Space data.

The images generated by the Ray-Space board need to be transferred to the 3D graphics board and composited in a scene rendered from 3D geometric data. The most effective way would be to put the Ray-Space generated images directly in the frame buffer of the 3D graphics board. Unfortunately, we could not find any satisfactory graphics accelerator which allows us to access the frame buffer. Instead, the generated images are transferred through the PCI bus of the host computer.

The Ray-Space Accelerator Board can be divided in two main blocks: the Ray-Space Engine and the Controller. The Ray-Space Engine is the heart of the board, and it is the actual hardware implementation of the Ray-Space algorithm. The Controller is based on a micro-controller and its main function is to load the Ray-Space data from the local hard disk into the memory of the Ray-Space Engine. The Controller does not participate directly in the rendering of the Ray-Space data.

In the following subsections, we describe in more details the Ray-Space Engine and the Controller.

4.1 Ray-Space Engine

The Ray-Space data for each object consists of the captured images of the object and the RS map information. The RS map information is calculated off-line so that the interpolation of the rays is not restricted to a hardware implementation and can be customized for different objects.

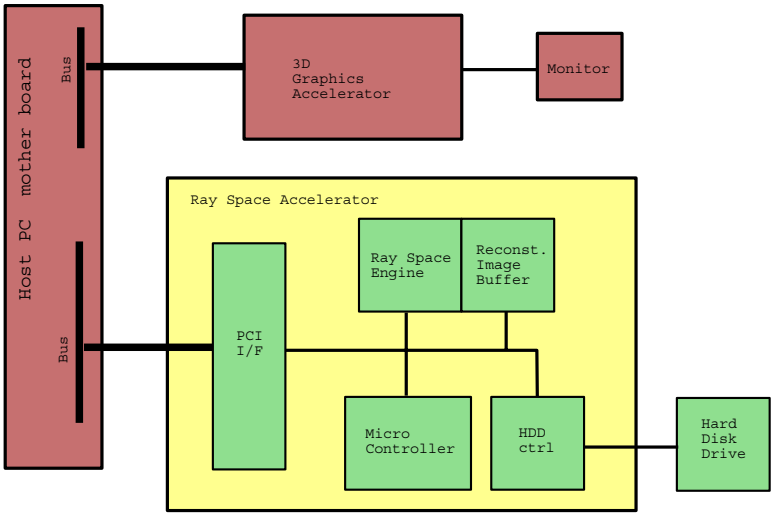


Figure 4 The Ray-Space Board.

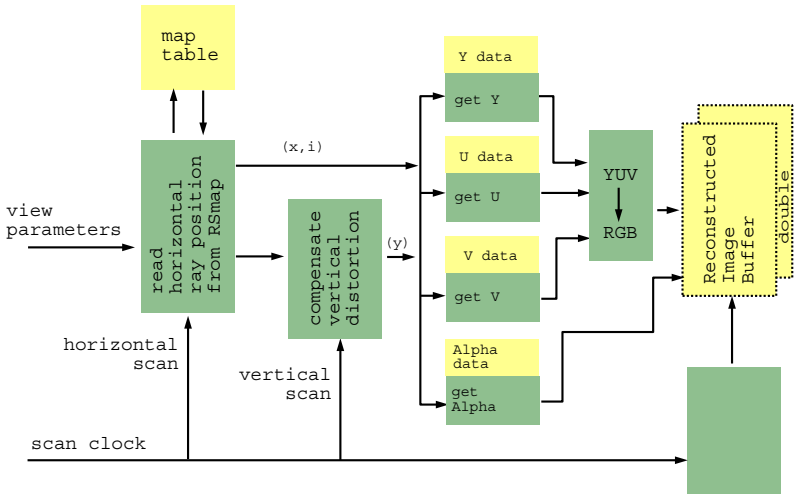


Figure 5 The Ray-Space Engine.

The architecture of the Ray-Space Engine is given in Figure 5. As already explained in Section 2, to reconstruct an image from the Ray-Space data, a straight line on the (x, u) plane is calculated. This line is determined by the view position and corresponds to a set of rays of the captured images.

To render an area of the display, a raster generator scans the area and each point in the display is mapped into a point in the straight line of the (x, u) space. When a valid ray position is found in the RS map, a captured image index and a column number (i, x) of the pixel to be fetched is returned. The RS map also gives the value used to compensate the horizontal distortion for each pixel. Using this value, the line position (y) of the required pixel can be calculated. The color (Y, U, V) and α values of the (i, x, y) pixel is then recovered from the captured image data set, converted to (R, G, B, α) format and put in the Reconstructed Image Buffer to store the view of the object. The Reconstructed Image Buffer is double buffered, so that a set of reconstructed images can be fetched by the host system while another set of images is being reconstructed.

Data Compression

Even though the Ray-Space method adopted is simplified to have only 2 degrees of freedom, it still requires a large number of images of the object to be rendered. For a more efficient use of the memory, it is desirable to compress the captured image data.

The images for the Ray-Space method are captured from an object rotating in a turntable. With this setup, images taken from close view angles tend to have high correlation. Several compression algorithm were suggested for a set of images with this characteristics [2], but for the Ray-Space method two features are essential when compressing the images: decoding speed and pixel random access.

The compression method chosen for the Ray-Space method uses a hybrid combination of motion-compensated prediction, Discrete Cosine Transform and Vector Quantization, as shown in Figure 6. From the image data set, some images are chosen as reference images and are saved uncompressed. To compress an image that is not a reference image, we divide it in blocks of 8x8 pixels. Each block is compressed independently using motion compensation with respect to a near reference image. The residual difference is processed and coded using DCT. The obtained DCT components are further coded by Vector Quantization. At this stage, the high frequency components of the DCT are eliminated. The transparency of the pixels are also coded by Vector Quantization. As shown in Figure 7, the decoding of the pixel values occurs in the inverse order.

This compression method has been implemented in software and used by the software implementation of the Ray-Space renderer. The compression ratio varies according to the objects' image information and the SNR tolerance. For a typical image data set consisting of 90 images where 10 images were used as references, the compression ratio has been around 1/10, with an SNR of 30.0 db.

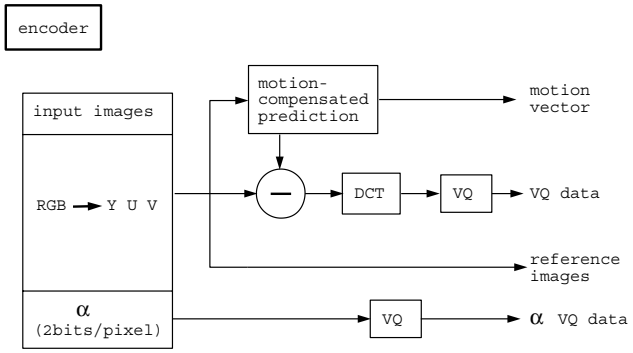


Figure 6 Image compression.

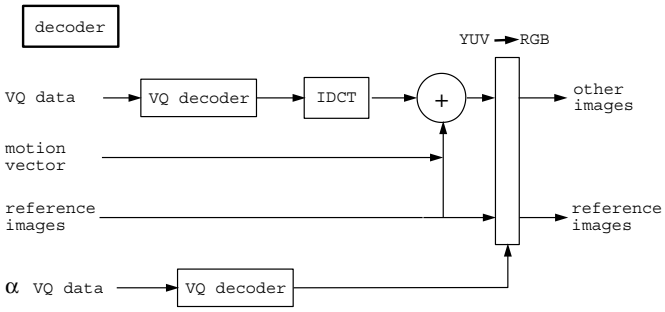


Figure 7 Image decompression.

4.2 Controller and Local Hard Disk

A typical application of the CyberMirage system uses several Ray-Space data. Since the amount of memory of the board is limited and might not hold the full data set, we split the virtual environment in several areas (e.g. showrooms). Each showroom is assigned a manageable number of Ray-Space objects that will be shown in that room. Using this partitioning, it is necessary to load only the Ray-Space data for the room where the viewer is. If the data is loaded into the Ray-Space board via the PCI bus, the performance is limited by the low bandwidth of the bus. To avoid the PCI bus bottleneck, a local hard disk is introduced in the architecture. A micro-controller is also added to control the access to the local hard disk. Its functions are to read the file, interpret its contents, and send the data to the memory of the Ray-Space engine.

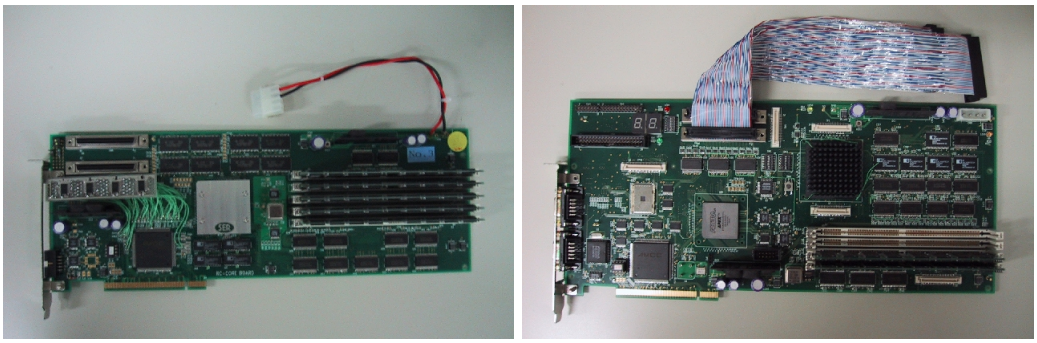
5 Hardware Implementation

We used one Altera EP20K400 FPGA chip, clocked at 40 MHz, to implement the following components:

- the Ray-Space Engine logic,
- the registers to store the parameter information of 20 Ray-Space objects,
- the registers to hold the view parameters to reconstruct 15 images in one rendering cycle.

The board has enough memory for 20 Ray-Space objects. Each object has a maximum of 120 captured images of at most 512 x 512 pixels. Furthermore, each object has a maximum of 12 reference planes whose maximum resolution is 1024 x 1024. The captured images are saved as (Y, U, V, α). The components U and V have one forth of the resolution (half the height and half the width) of the Y component. Unfortunately the decompression implementation logic could not fit inside the FPGA. The captured image data is therefore stored uncompressed. All things considered, the prototype was designed with 1 GBytes of on-board memory, including the memory necessary for the RS map data and the Reconstructed Image Buffer.

Due to the size of the FPGA chip, the on-board memory, and the micro-controller and its peripherals, the implementation was done in two PC cards: one for the Ray-Space Engine (Figure 8(a)), and another for the local micro-controller and hard disk interface (Figure 8(b)). The two boards are connected by a private local bus that allows a 20MB/s transfer rate of Ray-Space data.



(a) (b)

Figure 8 The RS Accelerator: (a) the Ray-Space engine board (b) the Controller board.

6 Software Library

The Ray-Space data module API is the interface between the application and the Ray-Space board driver. Figure 9 shows the software structure layers. The functions of the Ray-Space data module are:

- loading of Ray-Space data;
- management of the Ray-Space board memory and the data that it contains;
- pre-processing of the view parameters of the rendering command that is going to be issued;
- management of the rendering command queue;
- reading and saving the generated images from the Reconstructed Image Buffer;
- supplying the reconstructed image when requested by the application.

A software Ray-Space renderer, the Ray-Space Core Module, is also provided. It is used when the system needs to process Ray-Space data that the Ray-Space board can not hold, or to balance the load between the board and the host system. The other APIs, like OpenGL, are not modified since the Ray-Space modules can work independently from them.

The VRML loader was customized in order to accept the newly defined Ray-Space node structure. With this modified VRML, both geometric data and Ray-Space data can be described in common scene graphs.

7 Results and Discussion

With the current implementation of the prototype, we are able to reconstruct images from Ray-Space data at approximately 11 Mpixels per second. In other words, a 512 x 512 size image could be generated at more than 40 images/sec, or a scene with

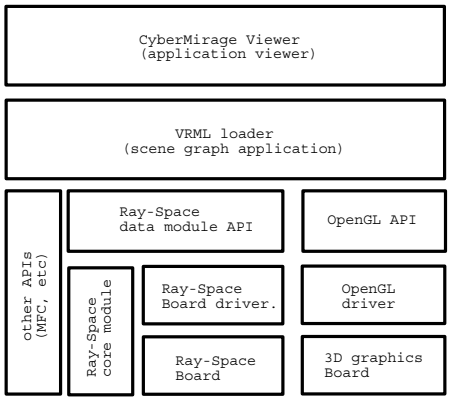


Figure 9 The Ray-Space driver.

15 images of 256 x 256 in size could be generated at approximately 11 fps.

In our implementation, we were restricted by the size of the FPGA logic. As a result, the current design has limited rendering performance. The main bottleneck in our implementation, however, is the transfer rate of the reconstructed images to the 3D graphic board. The access to the Image Buffer is done directly through the 32-bit 33MHz PCI bus. Even using the Bus Master transfer mode, our prototype cannot have the transfer rate of 44 Mbytes/sec necessary to transfer the images at the same speed as they were reconstructed.

We expect to cope with these problems with:

- a better implementation of the Ray-Space Engine in faster and denser logic circuits like ASIC to increase the image generation speed;
- multiple rendering pipelines running in parallel;
- faster memory and better memory access schemes;
- a more efficient implementation of the PCI bus interface;
- a PCI bus with a larger bandwidth like the PCI-X [11], which is a 64 bits, 133MHz interconnect

bus;

- a dedicated bus between the Ray-Space board and the 3D graphics card;
- compression of the reconstructed images before sending them to the graphics card.

In our opinion, these improvements would bring the performance of our hardware renderer to around 100 Mpixels per second.

Even though there is room for improvements in our prototype, the present implementation generates the images from Ray-Space data that are used in the CyberMirage, as shown in the Figures 10, 11 and 12. Figure 10 shows images rendered by the Ray-Space hardware. Figure 11 shows a typical scene of the CyberMirage, with the dolls and flowers rendered from Ray-Space data, and the shelves and walls rendered from polygons. The same scene, with the polygons rendered as wire frames is shown in Figure 12. Figure 11 has 9 images generated by the Ray-Space board and composited in the scene rendered by a Wildcat 4000 from a geometric model of approximately 10K polygons. Walkthroughs of this scene can be displayed at approximately 15 fps.

8 Conclusions

In this paper we described the development of a hardware Ray-Space renderer. We use this dedicated hardware in conjunction with a 3D graphics card in an interactive application to render photorealistic mages from both Ray-Space data and geometric model data.

Ray-Space representation allows to render images of objects with complex shapes and reflectance properties that would be hard to model with geometric primitives. Independently of the geometric complexity of the object, it takes approximately 30 minutes from capturing images of the object to view its Ray-Space data in a virtual scene.

We still have some issues to work out. The present implementation can be improved for speed and onboard memory usage, using better memory access schemes, faster logic, and the implementation of decompression for the captured image data. Moreover, the display of the reconstructed data needs to be improved, either by using a faster data bus to transfer the data from the Ray-Space board to the graphics accelerator board, or by implementing the geometric renderer in the same architecture of the Ray-Space renderer.

Currently we are exploring various applications for the CyberMirage system. In a few months we should complete another application: a virtual museum where the pieces are rendered from Ray-Space data captured from real relics.

References

[1] E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. In Landy and E. Movshon, editors, *Computational Models of Visual Processing*, pages 3–20. MIT Press, 1991.

[2] R. J. Clarke. *Digital Compression of Still Images and Video*. Academic Press, 1995.

[3] http://www.x-zone.canon.co.jp/CyberMirage/index-e.html.

[4] http://www.hc.t.u-tokyo.ca.jp/3D/CyberMirage/index-e.html.

[5] W. J. Dally, L. McMillan, G. Bishop, and H. Fuchs. The Delta Tree: An object-centered approach to image-based rendering. Technical Report AI memo 1604, MIT, May 1996.

[6] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54, 1996.

[7] A. Katayama, K. Tanaka, K. Oshino, and H. Tamura. A viewpoint dependent stereoscopic display using interpolation of multi-viewpoint images. In S. Fisher, J. Merritt, and B. Bolas, editors, *Stereoscopic Displays and Virtual Reality Systems II*, volume Vol. 2409 of *Proc. SPIE*, pages 11–20, 1995.

[8] M. Levoy and P. Hanrahan. Light Field Rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, 1996.

[9] T. Naemura, M. Kaneko, and H. Harashima. Orthographic approach to representing 3D images and interpolating light rays for 3D image communication and virtual environment. *Signal Process. Image Commun.*, 14:21–37, 1998.

[10] Y. Ohta and H. Tamura, editors. *Mixed Reality - Merging Real and Virtual Worlds*. Ohmsha, Ltd. & Springer-Verlag, 1999.

[11] http://www6.compaq.com/support/techpubs/whitepapers/tc990903tb.html.

[12] H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler. The VolumePro real-time ray-casting system. In *SIGGRAPH 99 Conference Proceedings*, pages 251–260, 1999.

[13] V. Popescu, J. Eyles, A. Lastra, J. Steinhurst, N. England, and L. Nyland. The WarpEngine: An architecture for the post-polygonal age. In *SIGGRAPH 2000 Conference Proceedings*, pages 433–442, 2000.

[14] M. Regan, G. Miller, S. Rubin, and C. Kogelnik. A real-time low-latency hardware light-field renderer. In *SIGGRAPH 99 Conference Proceedings*, pages 287–290, 1999.

[15] J. Toborg and J. Kajiya. Talisman: Commodity real-time 3D graphics for the PC. In *SIGGRAPH 96 Conference Proceedings*, pages 353–364, 1996.

[16] S. Uchiyama, A. Katayama, H. Tamura, T. Naemura, M. Kaneko, and H. Harashima. Cybermirage: Embedding ray based data in VRML world. In *Proc. VRAIS 97*, 1997.



Figure 10 Images rendered from Ray-Space data by the hardware Ray-Space renderer.



Figure 11 Geometric data and Ray-Space data composited in the same scene.

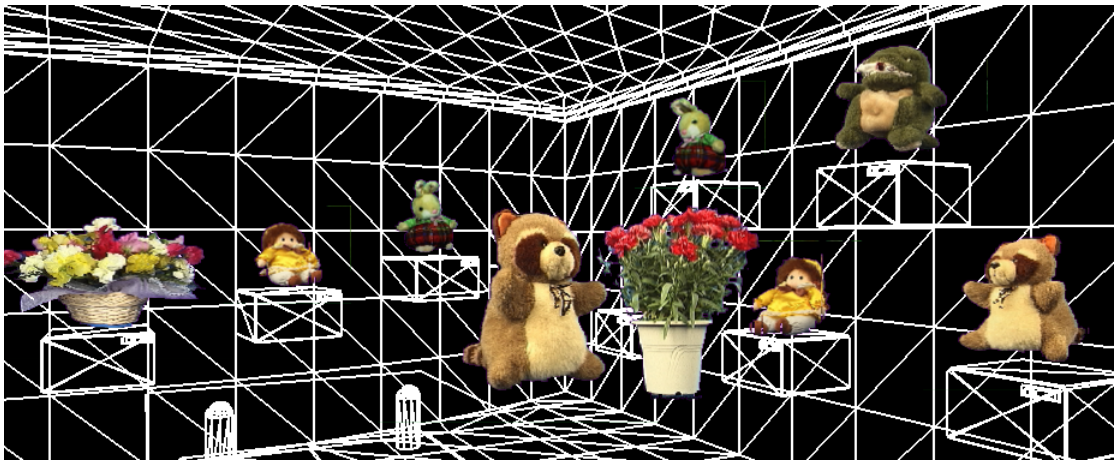


Figure 12 Geometric data and Ray-Space data - the 3D geometric primitives are shown as wire frame.