Photo-Realistic Driving Simulator using Eigen Texture and Real-Time Restoration Techniques by GPU

Ryo Sato[*1] Shintaro Ono[*2] Hiroshi Kawasaki[*1] Katsushi Ikeuchi[*2]
Saitama University[*1]
(255 Shimo-Ohkubo, Sakura-ku, Saitama 338-8570 JAPAN, +81-48-858-3855, {satou, kawasaki}@cgv.ics.saitama-u.ac.jp)
The University of Tokyo[*2]
(4-6-1 Komaba, Meguro-ku, Tokyo 153-8505 JAPAN, +81-3-5452-6242, {onoshin, ki}@cvl.iis.u-tokyo.ac.jp)

Computer modeling of a large-scale scene such as a city is an important topic in ITS, with applications in driving simulation, car navigation, city planning, etc. Image-Based Rendering (IBR) is an effective method for presenting a realistic scene, where real images are stored as a database in advance, and a novel view can be synthesized by cutting and stitching images in the database. However, the large size of the image database in IBR causes serious problems in actual applications, requiring the use of compression techniques. We propose a compression technique based on eigen images combined with a block-matching technique to get a better result. We also propose a technique to uncompress the data on the fly by using a Graphic Processing Unit (GPU), allowing us to perform high-speed rendering without raising the load on the CPU. This technique is also promising as a means of achieving realistic driving simulation systems.

Keywords: *Image compression, Driving Simulator, Image-Based Rendering, GPU, Omni-directional image*

1. Introduction

In recent years, research into computer modeling of large-scale scenes such as a city has been intensively conducted in many fields including ITS. For constructing and expressing such large scenes, Model-Based Rendering (MBR) and Image Based-Rendering (IBR) are mainly proposed.

MBR is a classical method that can easily reconstruct a virtual view from any arbitrary viewpoint by using explicit 3-D geometric model and texture information about the scene. Almost all traditional driving simulators are based on MBR architecture. However, reconstruction of the model of a large scene requires huge human cost and time, in spite of relatively low photo-reality. It is especially difficult to represent intricately shaped objects, such as trees with leaves, realistically by MBR, since such objects are hard to model with all texture information.

On the other hand, IBR is a method that reconstructs a virtual view by using a number of images captured beforehand. IBR can easily express the scene with high photo-reality even for intricately shaped objects, since images are used. A driving view generation system using IBR has been proposed [7] that can be created with far less human effort than one that uses MBR. However, one of the disadvantages of IBR is that it requires a large quantity of image data to render large scenes. Efficiently compressing the image data allows a realistic construction of a large scene while keeping the data size manageable.

In this paper, our purpose is to propose a system that efficiently compresses image data for IBR. IBR requires almost random access to a huge image database depending on view directions. In addition, since large-scene data is necessary for IBR, omnidirectional images that can capture the whole hemisphere in one shot are often used for efficiency. We noticed that the sequence of omnidirectional images captured along a road contains redundancies because the same object is captured from multiple directions. Exploiting this, we propose an effective compression method for the data by using an eigen texture method: applying principal component analysis (PCA) to particular image sequences [9].

This texture compression and restoration method is considered to be suitable for the feature of IBR that requires random access to a large image database. It is known that the eigen texture method can give a high compression ratio if the similarity among images in the image sequence is high, but gives a poor ratio if the sequence includes even subtle gaps caused by error. To improve the compression ratio, we propose accurate tracking by using block matching, based on Epipolar Plane Image (EPI) analysis [10].

The compressed image data in the eigen texture method can be restored by the product-sum operation of the eigen image with weighting coefficient, which requires the linear sum of every pixel. However, this processing becomes a problem in a case requiring real-time processing, such as a driving simulator. So we also propose a method to restore the compressed data using the GPU in which parallel processing is available, facilitating high-speed rendering without a high CPU

load. By using the proposed methods, we can realize real-time rendering with IBR on a standard PC and produce a realistic driving-view simulator.

We present the outline of this system in Section 2, compression in Section 3, rendering in Section 4, experiment in Section 5, and our conclusion in Section 6. In this paper, we refer to the driving-view generation system as the "driving simulator."

2. Outline of Realistic Driving Simulation System

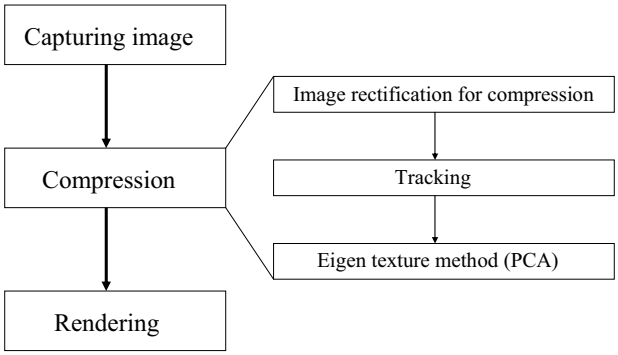


Figure 1. Overview of System

Figure 2. Omnidirectional image

Our system is composed of three processes: (1) Capturing omnidirectional images, (2) Compressing the images, and (3) Rendering an arbitrary scene from the compressed data (Figure 1).

We store sequential omnidirectional images, obtained by running along the actual road using an on-vehicle camera and shown in Figure 2, in an image database, and virtually generate a free-point view by cutting and stitching the image data based on [7].

The key points of this system are compression of texture for IBR and fast free-view point rendering using IBR. In the case of IBR, generally, quite a lot of image data is necessary for constructing a view even on scene. Moreover, since the omnidirectional image captures the spherical range at once, its data size becomes large. When a virtual viewpoint moves, it is required to load a vast quantity of image data sequentially into memory in a computer. This causes the problem that the drawing speed decreases because of the large CPU load and I/O transactions.

Therefore, we propose a method to compress images using high redundancy between adjacent frames on the omnidirectional image. We also need to consider the decompression algorithm to efficiently achieve real-time rendering.

3. Compression Method

The compression procedure is shown in the right area of Figure 1. After the simple image rectification, we first track to find redundancy, which later becomes subject to be compressed, inside the image sequence. Second, we compress the tracked partial images by using the eigen texture method [9].

If the purpose of compression is only for streaming playback, use of existing video compression technologies such as MPEG is suitable to achieve a high compression ratio, which is optimized for restoring a series of continuous images along a time line. In our case, however, as mentioned previously, a scene is rendered by restoring appropriate synthesized images depending on the viewpoint, using just a part of a huge image database, which is compressed. This is totally different from video stream compression, which cannot deal with such processing.

3.1. Image Rectification

Prior to the compression, omnidirectional images in the sequence are converted to perspective images viewing a lateral direction by simple coordinate conversion. That is, the images are projected onto a perpendicular plane along the side of the capturing course, as shown in Figure 3. By doing this, the transition of an object becomes linear, and appears as a straight line on an EPI, further described in 3.2.

Figure 3. Rectified image

3.2. Tracking

Tracking is preformed in order to find similar parts in the image sequence. The image sequence is omnidirectional, so an object viewed from multiple directions inside the sequence produces similar images. To find similarities, a block-matching method based on the correlative value between images is usually used. However, if objects move fast between adjacent frames, the searching path for block-matching becomes long and the result is usually unstable, especially for flat-colored blocks. Since this is the case in our input image sequence, we introduce another tracking method in advance of block-matching to retrieve a correct block sequence.

We propose a two-step method as follows: the first step is approximate global tracking, and the second step is precise tracking.

3.2.1. Global tracking by space-time image analysis

We take advantage of space-time image analysis for global tracking. By accumulating the image sequence in a temporal direction, a so-called space-time image volume can be constructed. An Epipolar Plane Image (EPI) is an image that appears on a planar cross-section of the space-time volume parallel to an epipolar plane among the sequential camera positions. In our case, we assume the plane to be a horizontal plane (Figure 4). This is true in an ideal case, where the pose of the capturing camera is upright and it moves on a horizontal plane. In actual cases this is not strictly true, but it is sufficient for our purpose since vibration of the camera is relatively small and can be dealt with by the later process.

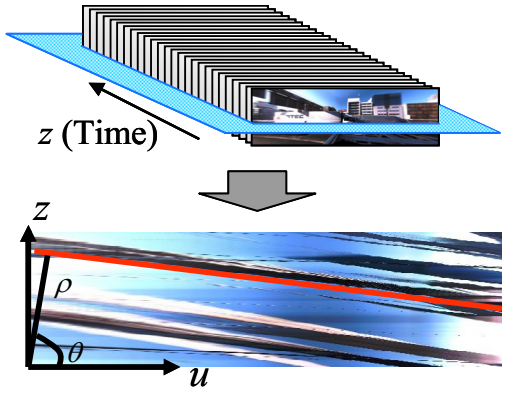


Figure 4. EPI

The trajectories of edge lines on the EPI are determined by camera's moving speed and the distance between the camera and the object. If the movement of the camera can be assumed as uniform, or can be roughly normalized as uniform by some external devices, the tracking process can simply be resolved into straight line detection on the EPI.

The tracking can be simplified in the case that the objects face on an approximately constant plane in the real world. A series of building facades in an urban scene is a typical example. In such a case, the global tracking can be expressed by one parameter: choosing a typical inclination among the straight lines in the EPI. Straight lines in the EPI can be detected by applying edge detection followed by a Hough transform. The detected line can be described as follows

$$\rho = u\cos\theta + z\sin\theta \qquad (1)$$

where the distance between the origin and the line is ρ, and the inclination angle between the perpendicular line and the *u*-axis is θ. Since the *z*-axis corresponds to time in the EPI, moving one pixel in the *z*-axis direction means changing one frame. Therefore, the amount of movement of the *u* direction of the object in one frame can be described as follows.

$$\frac{\partial u}{\partial z} = -\tan\theta \qquad (2)$$

This is the parameter of the global tracking.

3.2.2. Local Tracking by Block-Matching

In order to get a higher compression ratio, partial image sequences that are subject to be compressed should be as similar as possible. We perform local tracking to achieve this.

The process is shown in figure 5. The global motion of the object per frame towards the *u* direction can be calculated by Eq. (2). According to this, a block image drawn in bold at frame f_1 should move to the dotted block in frame f_2 and f_3. In the actual case, however, because of vibration and the minute change of the moving speed of the camera, the corresponding image blocks are shifted from the result of global tracking.

Therefore, the block image with the highest correlative value compared with the block image of the previous frame is searched by doing block matching between the frames around the result of the global tracking. The correlative value that is used by block matching uses a linear correlation coefficient.

Correlation coefficient *r* is as follows:

$$r = \frac{\sum_i \sum_j (x_{ij} - \bar{x})(y_{ij} - \bar{y})}{\sqrt{\sum_i \sum_j (x_{ij} - \bar{x})^2} \sqrt{\sum_i \sum_j (y_{ij} - \bar{y})^2}} \qquad (3)$$

where x_{ij}, y_{ij} are the pixel values of coordinates (*i, j*) of two arbitrary images (*X, Y*), and $\bar{x}$, $\bar{y}$ are those average values. The image with all the same pixel values ($x_{ij} = y_{ij}$ in all *i, j*) is 1.

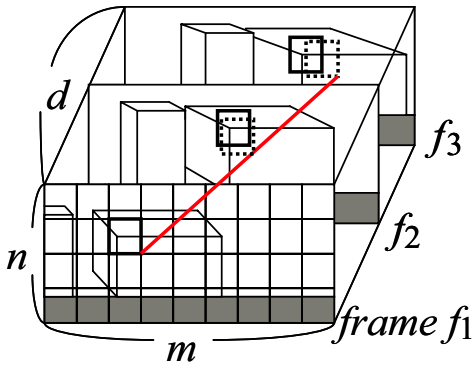


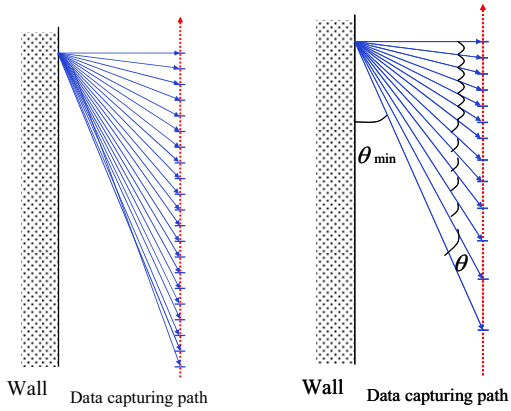
Figure 5. Local tracking by block matching

3.3. Sampling based on angle uniformity

A point in the real scene is viewed in various angles from a sequence of image capturing points. After the tracking, we can detect the "viewed angle" of the targeted points as Figure 6 (a).

Throughout all the tracked blocks among captured images, the distribution of the viewed angles is not uniform. Some of them become quite similar, especially when capturing points are far apart from the targeted point. Therefore, it is not efficient to treat all the capturing points equally and use all the blocks for compression.

To solve this problem, the blocks used for compression are sampled. Considering that synthesis of free-point view by IBR is based on stitching of spatial rays, the images are sampled uniformly using an angle between the assumed wall and viewpoint direction, θ (see figure 6). In addition, uniform sampling based on angle can avoid side effects of unstable tracking for $\theta \approx 0$ direction.



(a) Continuous image sequence (b) Uniform angle

Figure 6. Image sampling

3.4. Compression by eigen texture

The first video frame is divided into the unit of the block of m×n (Figure 5). For each block image, the tracking described in 3.2 and the sampling described in 3.3 are applied up to the *d*-th frame ($\theta = \theta_{\min}$); then, the sets of similar block images are obtained.

PCA is applied to each block set, which is compressed into *k* eigen images. Applying this to all m×n block sets, the whole image sequence is compressed.

At this time, a high compression ratio can be expected because the obtained image sets have high correlative value by using block matching. It is theoretically possible to apply general compression technologies such as MPEG for each block. However, they generally target sequential video and are not suitable for randomly accessing many frames, which is required in IBR.

4. Rendering

4.1. Restoration of Compressed Images

Restoration of the image uses the eigen image that is obtained by PCA, and the image of the original *i*-th frame X_i can be restored by the linear sum as follows:

$$X_i = \sum_{k=1}^{r} \left(w_{ik} * V_k \right) \qquad (4)$$

where *k*-th eigen image is V_k , weight coefficient of *i*-th frame is w_{ik} , and the number of principal components required in order to achieve a certain cumulative proportion is *r*.

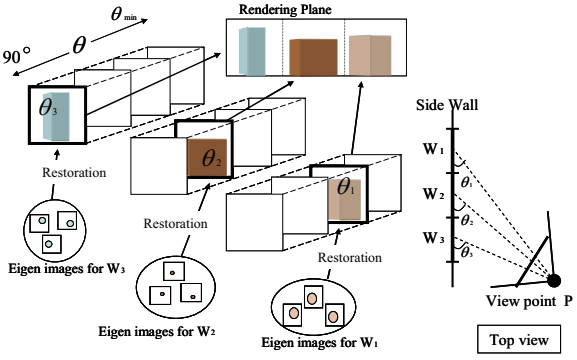
4.2. Rendering algorithm

4.2.1. Rendering side wall

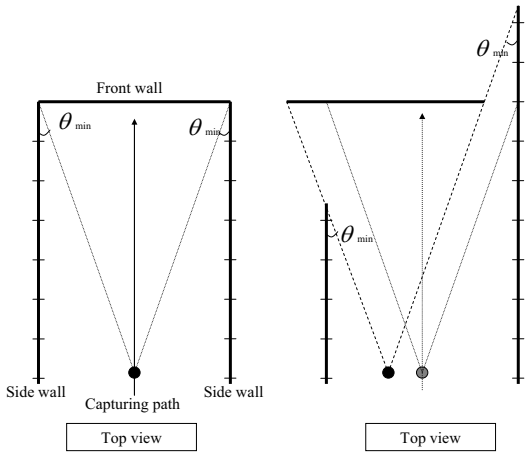
Figure 7 shows our method of rendering compressed data. Rendering at a certain viewpoint P is to select and map the texture corresponding to an angle composed of the viewpoint P and the center of the wall plane. The texture is restored from eigen images (compressed data) using the angle (θ_1 , θ_2 , θ_3 for each W_1 , W_2 , W_3 respectively). Since the angle θ changes dependent on the motion of viewpoint, it can generate the free viewpoint images.

4.2.2. Rendering the camera moving direction (front wall)

Since the image for which the viewing angle is less than threshold $\theta_{\min}$ is not sampled for compression, images cannot be recovered for that direction (it corresponds to the camera's moving direction). In our method, we directly use an omnidirectional image that was captured nearest from the viewpoint for the camera's moving direction. For rendering, we map the image on a front wall as shown in Figure 7 (b). The left figure shows that the viewpoint is exactly on the capturing path, and the right figure shows that the viewpoint moves to the left lane.



(a) Rendering side wall



(b) Arrangement of walls.

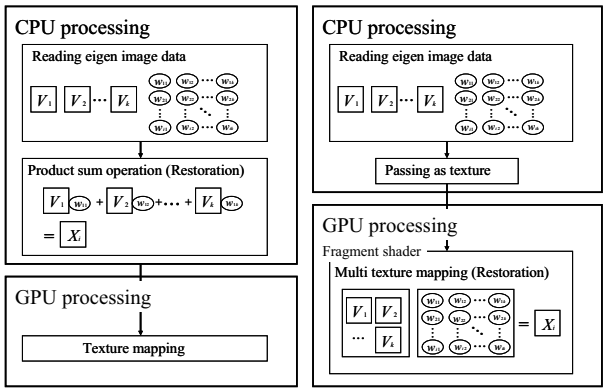
Figure 7. Rendering algorithm

4.3. Fast Restoration using GPU

We explain how to restore an original image X_1 using figure 8. From the viewpoint of computer architecture, first the compressed data (eigen images V_k and weight parameters w_{ik}) are loaded onto the main memory, and the product-sum operation of V_k and w_{1k} is processed using the CPU (Figure 8.(a)). The image obtained is then passed to the GPU (a core chip in graphics hardware) as a texture, and it is used for texture mapping. This is an ordinary CPU-based restoration process. However, when restoring the original image X_2, second, a procedure similar to w_{2k} is needed in a weighting coefficient, and processing takes time. The complexity of this method is $O(n^2)$, so the load on the CPU increases rapidly if the texture size becomes large.

On the other hand, graphics hardware is improving in performance recently, and it is now used for multiple purposes. In our case, we propose to restore the original image by using the GPU. After the compressed data have been read using the CPU, the data are passed to the GPU as textures, and a product-sum operation is processed only in the fragment shader of the GPU (figure 8.(b)). Even when restoring the original image X_2, the data can be processed by the product-sum operation of V_k and w_{2k}, which have been held beforehand, and it is not necessary to remake a texture. So this permits restoration of an original image without increasing the load on the CPU. And calculation load can be set constant even if the texture size becomes large.

We implemented product-sum operation on the fragment shader of the GPU as a multi texture technique, which can store several textures for the same polygon. We used the eigen image texture, the weight coefficient texture, and the average texture as the multi texture (Figure 9). In eigen image texture, eigen image from V_1 to V_k is arranged to the *u* coordinate in order. In weight coefficient texture, the weight coefficient corresponding to V_k from w_{1k} to w_{ik} is arranged in the same way. In average texture, the average value from 1 to *i* is arranged in the same way. So the images are restored by calculating the product-sum of Eq. (4) for each pixel, using multiple textures. When the view direction is changed, an image can be restored correctly by retrieving a weight from the weight texture.



(a) CPU-based (b) GPU-based

Figure 8. Image restoration process

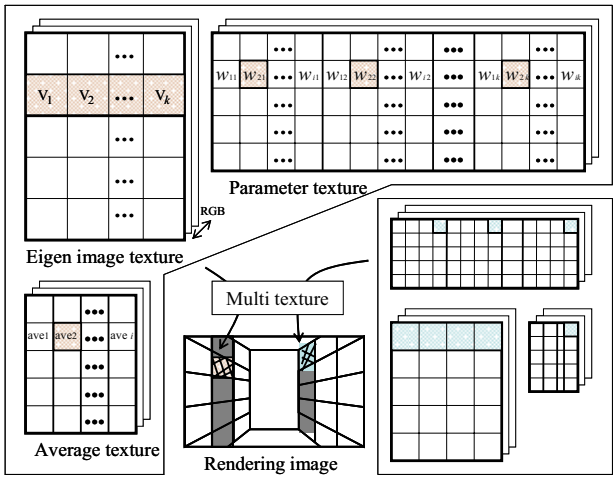


Figure 9. Implementation of multi-texture mapping

5. Experiment

5.1. Acquisition of omnidirectional image

Many of the applications of IBR make use of omnidirectional images because a wide field of view is available [2-4]. Omnidirectional images are usually acquired in one of two ways. The first uses a rotationally symmetric mirror and a single camera [5], and the second uses multiple cameras and merges them [6].

The rotationally symmetric mirror and a single camera have the advantage that the 360 degree horizontal panoramic image is captured in a single shot. However, it is difficult to obtain a sufficient resolution with current CCDs for the realistic rendering of a city.

On the other hand, omnidirectional imaging systems using multiple cameras, which currently improve the direction problems, can solve the problem [6, 8]. Therefore, we use a multiple camera-based system (Figure 10) to capture the texture data. The image is captured using synchronized CCDs, so is temporally consistent.

Figure 10. Capturing camera

5.2. Result of Tracking and Compression

Each of the omnidirectional images is divided into a unit of 16×16, and the experiment of compression is done using the image row of 70 frames (sampled θ = 1 degree, θ_{min} = 20 degree). Here, let us describe only global tracking using EPI as T1, and both global and local tracking using EPI and block matching as T2.

In Figure 11, (a1), (b1) are the results of the block images obtained by T1, and (a2), (b2) are those by T2. It turns out that the objects vibrate after T1, but not after T2.

Figure 12 shows eigen values in descending order in the two cases of tracking, T1 and T2. It shows that T2 exceeds T1, regarding eigen values corresponding to the 1st, 2nd, and 3rd principal components, especially for the 1st principal component. This quantitatively ensures that more similar block images were able to be tracked by using block matching.

Figure 13 shows the cumulative proportion of eigen values against component images. The graph based on T2 has risen more quickly than the graph based on T1, meaning that the original image is reconstructed by fewer images.

Figure 14 shows PSNR (peak signal-to-noise ratio) for various numbers of eigen images. It turns out that using T2, a high-quality image can be restored by fewer images compared with T1. It also proves that 40dB of PSNR, which is generally recognized as providing enough quality in video compression, is achieved by using 30% numbers of eigen images.

Table 1 shows the final results of these compressions. As a result, it turns out that the compression ratio is improved.

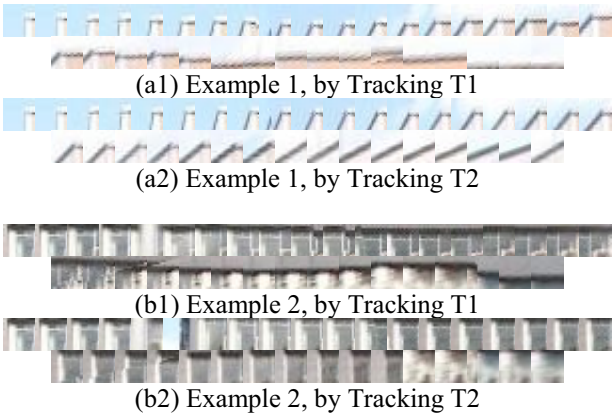


Figure 11. Block images resulting from global tracking using EPI analysis (T1), and from global and local tracking using block matching after EPI analysis (T2).

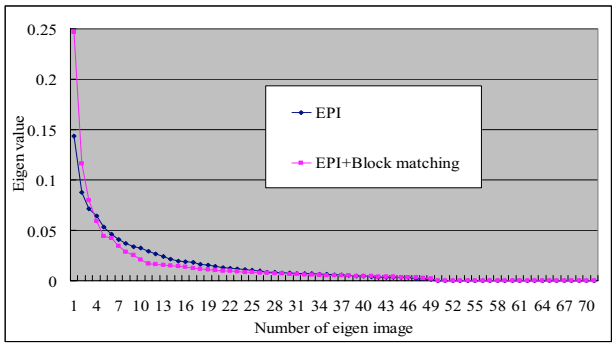


Figure 12. Eigen value

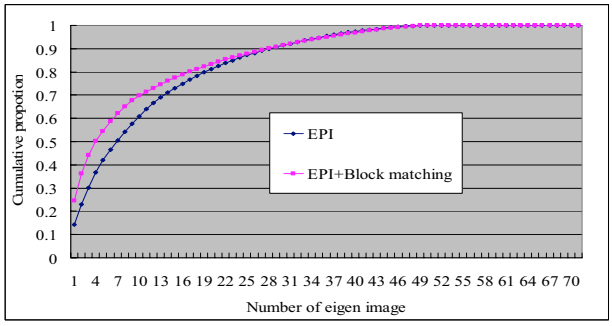


Figure 13. Cumulative proportion

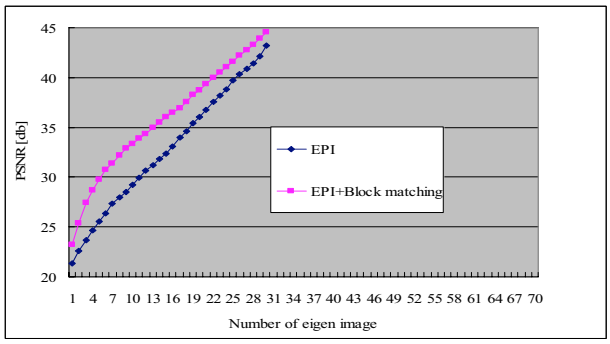


Figure 14. PSNR

Table 1. Result of compression

Example 1	(T1) EPI	(T2) EPI+Blockmaching
Average of Correlative value	0.756491	0.936779
Cumulative proportion 50%	7.3	4.3
(Compression ratio of data)	28.1/280KB	16.6/280KB
Cumulative proportion 70%	14.0	10.6
(Compression ratio)	53.6/280KB	40.9/280KB
Cumulative proportion 90%	28.6	28.3
(Compression ratio)	109/280KB	108/280KB
Example 2	EPI	EPI+Blockmaching
Average of Correlative value	0.842592	0.929996
Cumulative proportion 50%	7.0	5.0
(Compression ratio)	26.7/280KB	19.1/280KB
Cumulative proportion 70%	15.0	12.0
(Compression ratio)	57.3/280KB	45.8/280KB
Cumulative proportion 90%	30.3	28.0
(Compression ratio)	115/280KB	107/280KB

5.3. Comparison in Processing Time for Rendering

The processing time for rendering by the proposed method was compared with that of the conventional method. The comparison is performed by measuring the time of processing that repeats the process of restoring the original 70 images sequentially 100 times. The PC used is Intel Core 2 (2.66GHz), and the GPU is Quadro FX 550.

The result is the graph of figure 15. The horizontal axis shows image size and the vertical axis shows processing time (average of 10-times trial). Both rendering methods took approximately the same processing time when the image size was smaller than 64 × 64 or less, but the processing time using the conventional method (CPU) increased rapidly as the image size became larger.

On the other hand, the proposed rendering method kept constant processing time regardless of the image size. Therefore, by the rendering method using the GPU, it was shown that high speed and stable rendering are realizable.

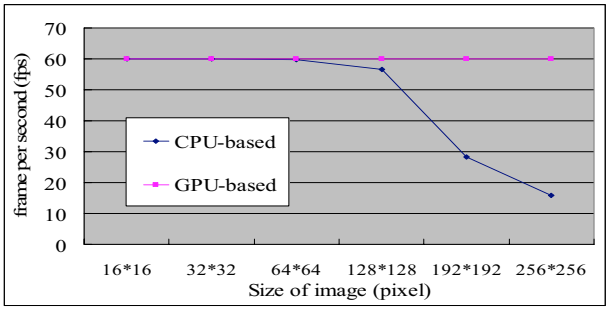


Figure 15. Comparison of processing time

5.4. Result of rendering

The rendering of a large-scale scene was done actually by the rendering algorithm described in 4.2. Figure 16 (a), (b), (c) are the results of rendering, which are the scenes from viewpoints (a), (b), (c) respectively in Figure 16 (d). These scenes are rendered by sum of eigen images under 75% cumulative proportion.

5.5. Experimental Evaluation by Test Subjects

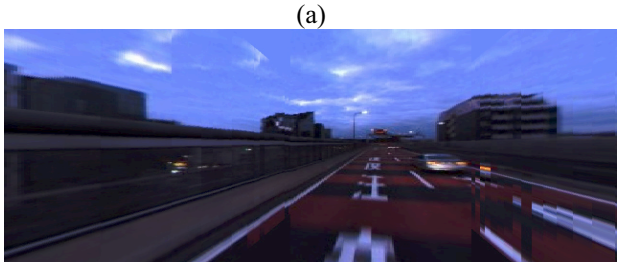
To confirm whether there exist any serious influences in image compression, we performed an experimental evaluation by test subjects. We showed two videos, one a simple raw image sequence as shown in Figure 2, and the other a rendered image sequence generated by our methods (For convenience, two videos are again encoded by MPEG1 with same bit rates). We asked our subjects how different two videos are as driving scenes, and got the result as Table 2.

In spite of the fact that we showed original raw images as comparative counterparts of compressed images, there were 5 persons out of 15 who replied that they were “different.” No one replied that they were “totally different.”

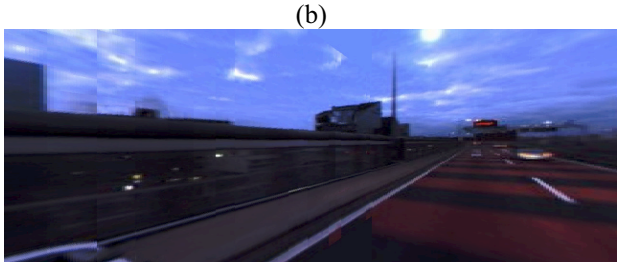
Table 2. Result of evaluation experiment

Evaluation result	person
Totally identical	2
Similar	4
Undecided	4
Different	5
Totally different	0

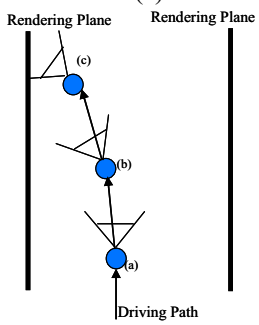
(a)



(b)



(c)



(d) Viewpoints of the scenes

Figure 16. Result of rendering

6. Conclusion

In this paper, we proposed two technologies for a photo-realistic driving simulator. One is the novel compression method for omnidirectional images by using eigen textures, and the other is a high-speed rendering algorithm using the GPU.

Since compression ratio becomes low when there is a translational difference among a series of image blocks to be compressed, we proposed a tracking method using block matching based on EPI analysis, and improved the compression ratio.

The compression data can be easily restored by a product-sum operation of the eigen images and the weight coefficient. Taking advantage of the fact that this operation is totally linear, we implemented the operation using a fragment shader in our graphics hardware, and achieved more high-speed and stable rendering without raising the load of CPU.

By this system, a photo-realistic driving simulator can be achieved.

References

[1] J. Oike, T. Oo, H. Kawasaki, and Y. Ohsawa, “Compression of View Dependent Texture for Omnidirectional Image (in Japanese)”, *Forum on Information Technology (FIT)*, Sep. 2004.

[2] C. J. Taylor, “Video plus”, *IEEE Workshop on Omnidirectional Vision*, pp. 3-11, 2000.

[3] H. Kawasaki, K. Ikeuchi, and M. Sakauchi, “Light field rendering for image-scale scenes”, *Proc. International Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. 2, pp. 64-71, Kauai, Hawaii, US, 2001.

[4] M. Hori, M. Kanbara, and N. Yokoya, “Novel Stereoscopic View Generation by Image-Based Rendering Coordinated with Depth Information (in Japanese)”, *IEICE Technical Report*, PRMU2006-185, 2007.

[5] Y. Onoue, K. Yamasawa, H. Takemura, and N. Yokoya, “Telepresence by realtime view-dependent image generation from omnidirectional video streams”, *Computer Vision and Image Understanding (CVIU)*, pp. 154-165, 1998.

[6] T. Mikami, T. Oo, S. Ono, H. Kawasaki, Y. Osawa, K. Ikeuchi, “Distortion-Free Fusion of Multiple Video Camera Images Using EPI Analysis”, *IEICE Transactions on Information and Systems*, Vol. J89-D, No. 6, pp. 1336-1347, Jun. 2006.

[7] S. Ono, K. Ogawara, M. Kagesawa, H. Kawasaki, M. Onuki, K. Honda, and K. Ikeuchi, “Development of Photo-Realistic and Interactive Driving View Generator by Synthesizing Real Image and Artificial Geometry Model”, *International Journal of ITS Research*, Vol. 3, No. 1, pp.19-27, Nov. 2005.

[8] Point Grey Research Inc, “Ladybug2”, http://www.ptgrey.com

[9] K. Nishino, Y. Sato, K. Ikeuchi, “Eigen-texture method: Appearance Compression based on 3D Model”, *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 618-624, Jun. 1999.

[10] R. Bolles, H. Baker and D. Marimont: “Epipolar plane image analysis: an approach to determining structure from motion”, *Int. J. of Computer Vision*, Vol. 1, pp. 7–55, 1987.

Ryo Sato received the BE degree from Saitama University in 2007. Currently he is a master student in Graduate School of Science and Engineering in Saitama University.

Shintaro Ono received the BE degree in 2001 and PhD degree in 2006 from The University of Tokyo. Currently he is a Project Research Associate in Collaborative Research Center for Advanced Mobility (ITS Center), The University of Tokyo, and Joint Research Staff in Saitama University. His research interests include computer vision/graphics and sensing system for ITS, and digital archiving of cultural heritage objects.

Hiroshi Kawasaki received the Ph.D. degree in Information and Communication Engineering from University of Tokyo, Japan, in 2003. He started working at Saitama University in 2003. Prior to Saitama University, he worked at Microsoft Research Redmond as an internship in 2000. His current research focus is on capturing a shape and texture and rendering them photo-realistically in the computer for ITS and VR/MR systems. He has published over 40 research papers including CVPR, IJCV, 3DIM, Eurographics and MVA in computer vision, computer graphics and ITS and won several awards including Songde Ma Outstanding Paper Award (best paper for ACCV) in 2007.

Katsushi Ikeuchi received the BE degree from Kyoto University in 1973 and the PhD degree from the University of Tokyo in 1978. After working at the AI Lab in MIT, Electrotechnical Laboratory in MITI, and the School of Computer Science in CMU, he joined the University of Tokyo in 1996, and is currently a full professor. His research interest spans computer vision and graphics, virtual reality, robotics, and ITS. In these research fields, he has received several awards, including the David Marr Prize and IEEE R&A K-S Fu memorial best transaction paper award. In addition, in 1992, his paper was selected as one of the most influential papers to have appeared in the AI Journal within the past 10 years. His activities include general chair, IROS95, ITSC00, IV01, IV06; program chair, CVPR96, ICCV03; Associate Editor, IEEE TRA, IEEE TPAMI; distinguished lecture SPS (2000-2002) , RAS (2004-2006). He was elected as an IEEE fellow in 1998. He is the EIC of the International Journal of Computer Vision.

- *Received date: March 21, 2008*
- *Received in revised form: September 8, 2008*
- *Accepted date: October 30, 2008*
- *Editor: Shinji Ozawa*