Real-Time Video-Based Rendering from Multiple Cameras

Vincent Nozick Hideo Saito

Graduate School of Science and Technology,
Keio University, Japan
E-mail: {nozick,saito}@ozawa.ics.keio.ac.jp

Abstract

In recent years, many Image-Based Rendering techniques have advanced from static to dynamic scenes and thus became Video-Based Rendering (VBR) methods. But actually, only few of them can render new views in live. We present a new VBR system that creates new views of a dynamic scene in live. This system provides high quality images and does not require any background extraction. Our method follows a plane-sweep approach and reaches real-time rendering using consumer graphic hardware (GPU). Only one computer is used for both acquisition and rendering. The video stream acquisition is performed by at least 3 webcams. We propose an additional video stream management that extends the number of webcams to 10 or more. These considerations make our system low-cost and hence accessible for everyone.

1 Introduction

Given several video streams of the same scene, Video-Based Rendering (VBR) methods provide new views of that scene from new view points. VBR is then an extension of Image-Based Rendering that can handle dynamic scenes.

In recent years, most of the proposed VBR techniques focused on the visual quality rather than on the computation time. To achieve this purpose, they use a large amount of data and sophisticated algorithms that prevent them from live rendering. Therefore, the video streams are recorded to be computed off-line. The rendering step can begin only when the scene information has been extracted from the videos. Such three-step approaches (record - compute - render) are called *off-line* since the delay between acquisition and rendering is long in regard to the final video length. *On-line* methods are fast enough to extract information from the input videos, create and display a new view several times per second. The rendering is then real-time but also live.

In this article, we present a new VBR method that creates new views of the scene on-line. Our method does not require any background extraction and therefore is not limited to render a unique object. This method provides good quality new views using only one computer. Most of out tests were computed from 4 or more webcams connected to a laptop. Hence, this method is low-cost and compatible with most consumer device configuration.

We also propose an additional video stream management to increase the number of cameras to 10 or more. Only the 4 most appropriate cameras are used to compute the new view. This technique extends the range of available virtual view points and improves the visual result.

2 Previous work

This section surveys previous work on both recent off-line and on-line Video-Based Rendering techniques.

2.1 Off-line Video-Based Rendering

The first proposed VBR method is the Virtualized Reality presented by Kanade et al. [5]. The video streams are first recorded from 51 cameras. Then every frame of every camera is computed to extract a depth map and create a reconstruction. Considering the amount of data, this precomputing step can be long. Finally, the new views are computed from the reconstruction of the most appropriate cameras.

Goldlucke et al. [3] and Zitnick et al. [14] follow the same approach. Goldlucke et al. use 100 cameras and create new views of the scene in real-time. Zitnick et al. provide hight quality images in real-time using 8 cameras. The depth maps are computed using a segmentation method and the rendering is performed with a layered image representation. The Stanford Camera Array presented by Wilburn et al. [10] computes an optical flow instead of a depth-map and provides real-time rendering from 100 cameras. Franco and Boyer [2] provide new views from 6 cameras with a Visual Hulls method.

Considering the large amount of data or the time consuming algorithms used by these previous methods, they appear to be hardly adaptable to on-line rendering.

2.2 On-line Video-Based Rendering

Only few VBR methods reach on-line rendering. Powerful algorithms used for off-line methods are not suited for real-time implementation. Therefore, we can not expect from on-line methods the same accuracy provided by off-line methods.

The most popular on-line VBR method is the Visual Hulls algorithm. This method extracts the silhouette of the main object of the scene on every input image. The shape of this object is then approximated by the intersection of the projected silhouettes. There exist several on-line implementations of the Visual Hulls described in [7]. The most accurate on-line Visual Hulls method seems to be the Image-Based Visual Hulls presented by Matusik et al. [8]. This method creates news views in real-time from 4 cameras. Each camera is controlled by one computer and an additional computer create the new views. The method proposed by Li et al. [6] is probably the easiest to implement. The main drawback of the Visual Hulls methods is the impossibility to handle the background of the scene. Hence, only one main "object" can be rendered. Furthermore, the Visual Hulls methods usually require several computers, which makes their use more difficult.

Another possibility to reach on-line rendering is to use a distributed Light Field as proposed by Yang et al. [11]. They present a 64-camera device based on a client-server scheme. The cameras are clustered into groups controlled by several computers. These computers are connected to a main server and transfer only the image fragments needed to compute the new view requested. This method provides real-time rendering but requires at least 8 computers for 64 cameras and additional hardware.

Finally, some plane-sweep methods reach on-line rendering using graphic hardware (GPU). The plane-sweep algorithm introduced by Collins [1] was adapted to on-line rendering by Yang et al. [12]. They compute new views in real-time from 5 cameras using 4 computers. Geys et al. [4] also use a plane-sweep approach to find out the scene geometry and render new views in real-time from 3 cameras and one computer. Since our method belongs to the latter family, we will expose the basic plane-sweep algorithm and [12, 4] contribution in the next section. Then we will detail our method.

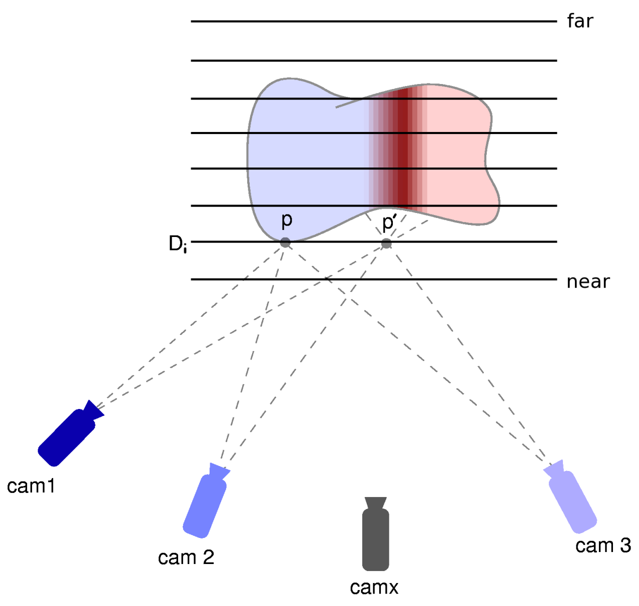

Figure 1. Plane-sweep : guiding principle.

3 Plane-Sweep Algorithm

This section expose the basic plane-sweep algorithm and surveys existing implementations.

3.1 Overview

The plane sweep algorithm provides new views of a scene from a set of calibrated images. Considering a scene where objects are exclusively diffuse, the user should "place" the virtual camera cam_x around the real video cameras and define a $near$ plane and a far plane such that every object of the scene lies between these two planes. Then, the space between $near$ and far planes is divided by parallel planes D_i as depicted in Figure 1.

Consider a visible object of the scene lying on one of these planes D_i at a point p. This point will be seen by every input camera with the same color (i.e. the object color). Consider now another point p' lying on a plane but not on the surface of a visible object. This point will probably not be seen by the input cameras with the same color. Figure 1 illustrates these two configurations. Therefore, the plane sweep algorithm is based on the following assumption : a point lying a plane D_i whose projection on every input camera provides a similar color potentially corresponds to the surface of an object.

During the new view creation process, every plane D_i is computed in a back to front order. Each pixel p of a plane D_i is projected onto the input images. Then, a score and a representative color are computed according to the match-

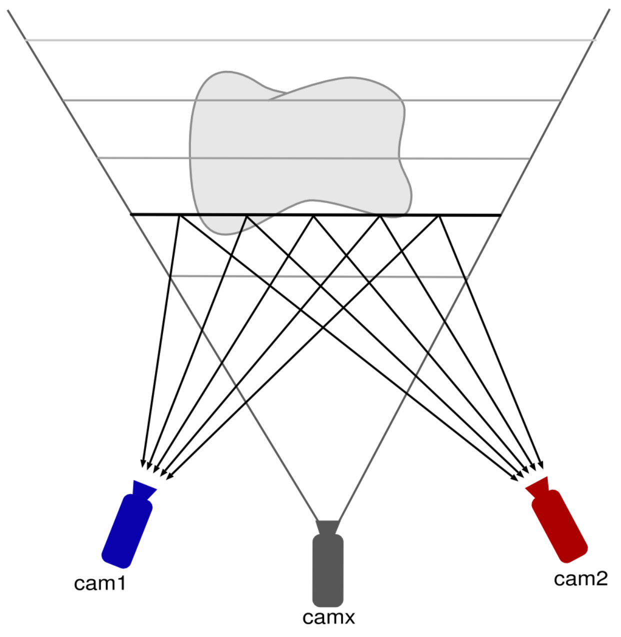

Figure 2. Every point of the current plane is projected on the input images. A score and a color are computed for these points according to the matching of the colors found.

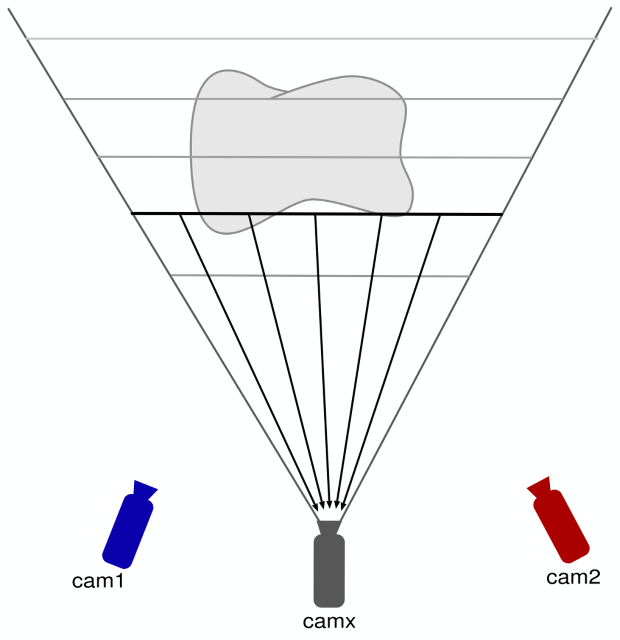

Figure 3. The computed scores and colors are projected on the virtual camera.

ing of the colors found. A good score corresponds to similar colors. This process is illustrated on Figure 2. Then, the computed scores and colors are projected on the virtual camera cam_x. The virtual view is hence updated in a z-buffer style : the color and score (depth in a z-buffer) of pixel of this virtual image is updated only if the projected point p provides a better score than the current score. This process is depicted on Figure 3. Then the next plane D_{i+1} is computed. The final image is obtained when every plane is computed.

3.2 Previous Implementations

Yang et al. [12] propose an implementation of the plane-sweep algorithm using register combiners. The system choose a reference camera that is closest to cam_x. During the process of a plane D_i, each point p of this plane is projected on both the reference image and the other input images. Then pair by pair, the color found in the reference image is compared to the color found in the other images using a SSD (Sum of Squared Difference). The final score of p is the sum of these SSD.

This method provides real-time and on-line rendering using 5 cameras and 4 computers, however the input cameras have to be close to each other and the navigation of the virtual camera should lie between the viewpoints of the input cameras, otherwise the reference camera may not be representative of cam_x. Lastly, moving the virtual camera may change the reference camera and induce discontinuities in the computed video during this change.

Geys et al.'s method [4] begins with a background extraction. The background geometry is supposed to be static. This assumption restricts the application of the plane-sweep algorithm to the foreground part. The scoring method used is similar to the method proposed by Yang et al. but they only compute a depth map. Then, an energy minimization method based on a graph cut algorithm (CPU) cleans up the depth map. A triangle mesh is extracted from the new depth map and view dependent texture mapping is used to create the new view. This method provides real-time and on-line rendering using 3 cameras and only one computer. However, the background geometry must be static.

4 Our Scoring Method

Our main contribution to the plane sweep algorithm concerns the score computation. Indeed, this operation is a crucial step since both visual results and speedy computation depend on it. Previous methods computes scores by comparing input images with the reference image. We propose a method that avoids the use of such reference image that may not be representative of the virtual view. Our method also use every input image together rather than to compute images by pair.

Since the scoring stage is performed by the graphic hardware, only simple instructions are supported. Thus a suitable solution is to use variance and average tools. During the process of a plane D_i, each point p of D_i is projected on every input image. The projection of p on each input image j provides a color c_j. The score of p is then set as the variance of the c_j. Thus similar colors c_j will provide a small variance which corresponds to a high score. On the contrary, mismatching colors will provide a high variance corresponding to a low score. In our method, the final color of p is set as the average color of the c_j. Indeed, the average of similar colors is very representative of the colors set. The average color computed from mismatching colors will not be a valid color for our method however, since these colors also provide a low score, this average color will very likely not be selected for the virtual image computation. This plane sweep implementation can be summarized as follows :

```
○ reset the scores of the virtual camera
○ **for** each plane $D_i$ from *far* to *near*
    • **for** each point (fragment) $p$ of $D_i$
        → project $p$ on the $n$ input images.
          $c_j$ is the color obtained from this projection on the $j^{th}$ input image
        → compute the color of $p$ :
          $color_p = \frac{1}{n} \sum_{j=1...n} c_j$
        → compute the score of $p$ :
          $score_p = \sum_{j=1...n} (c_j - color)^2$
    • project all the $D_i$'s scores and colors on the virtual camera
    • **for** each pixel $q$ of the virtual camera
        → **if** the projected score is better than the current one
          **then** update the score and the color of $q$
○ display the computed image
```

This method does not require any reference image and all input images are used together to compute the new view. The visual quality of the computed image is then noticeably increased. Moreover, this method avoids discontinuities that could appear in the virtual video when the virtual camera moves and changes its reference camera. Finally, this method is not limited to foreground objects.

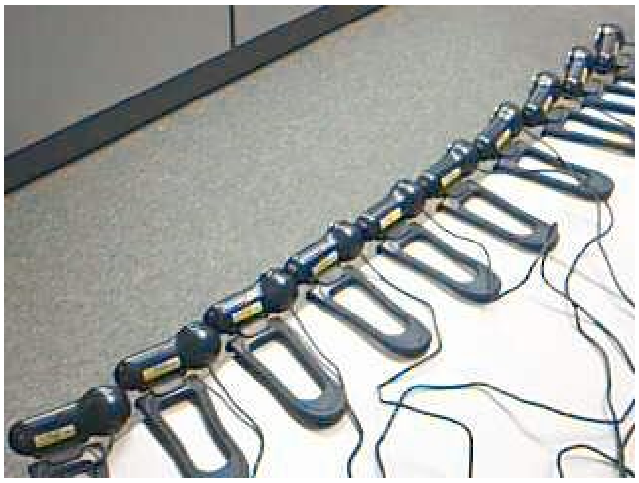

Figure 4. 10 webcams connected to a laptop via a usb hub.

5 Camera Array

A limitation of the plane-sweep method is the location of the input cameras : they need to be close to each other to provide engaging new virtual views. In fact, the closer they are, the better the final result is. The problem is then how to extend the range of available virtual view points without any loss of visual quality. Real-time plane-sweep method is limited in the number of input images used since the score computation time linearly depends on the number of input images. Furthermore, real-time video streams control requires special devices when the number of cameras increases too much.

We propose a webcam management to handle up to 10 or more usb cameras from a single computer (Figure 4). Considering the position of the virtual camera, the system selects the 4 most appropriate cameras. Only these cameras are used to compute the new view. The video streams from non-selected cameras are disabled. Then, for the next view, if the virtual camera moves, the set of selected input cameras is updated. Concerning the cameras configuration, every disposition is acceptable since the cameras are no too far from each other and are placed facing the scene. In such configurations, the most appropriate cameras to select for the new view computation are the nearest ones from the virtual camera. This method does not decrease the video stream acquisition frame rate since no more than 4 webcams are working at the same time.

This method can be used to extend the range of available virtual view points or just to increase the visual quality of the new views by using a dense cameras disposition. In a circle arc configuration, using 8 webcams rather than 4 will cover 60° instead of 30°. If the user prefers to place the additional cameras in the 30° area, then the visual quality of the created views will highly increase.

Finally, we believe that even with 5 or 10 additional webcams, this system remains low-cost since the price of a webcam is much lower than the price of additional computers required by others methods.

6 Implementation

Since our webcams have a fixed focal length, we compute accurately their internal parameters using Zhang calibration [13]. Then we can freely move them for our experimentations and only a single view of a calibration chessboard is required to perform a full calibration. If every webcam is of the same model, it is possible to assign to them the average internal parameter matrix. Even if we do not recommend that procedure, it can be very useful if the user needs to unplug and replug the webcams. In that case, the system cannot identify the webcams and attribute them the right internal parameter matrix. Color calibration can be performed by the method proposed by Magnor [7, page 23]. This method is effective only for small corrections.

We usually set the far plane as the calibration chessboard plane. The user should then determine the depth of the scene to define the $near$ plane. These two planes can also be set automatically using a precise stereo method as described in Geys et al. [4].

We use OpenGL for the rendering part. For each new view, we propose a first optional off-screen pass for every input image to correct the radial distortion and the color using Frame Buffer Objects. Implementation indications can be found on [9]. This pass is optional since some webcams already correct the radial distortion. Color correction is required only if auto-brightness can be disabled.

Each plane D_i is drawn as textured `GL_QUADS`. The scoring stage is performed thanks to fragment shaders. First, D_i's points (fragments) are projected onto the input images using projective texture mapping. The texture coordinates are computed from the projection matrices of each input camera. Multitexturing provides an access to every texture simultaneously during the scoring stage. Then, this fragment program computes each score and color using the algorithm described in section 4. The scores are stored in the `gl_FragDepth` and the colors in the `gl_FragColor`. Then we let OpenGL select the best scores with the z-test and update the color in the frame buffer. To compute a depth-map rather than a new view, we just set the `gl_FragColor` to the `gl_FragCoord.z` value. Most of the computation is done by the graphic card, hence the CPU is free for the video stream acquisition and the virtual camera control.

7 Results

We tested our method on a laptop core duo 1.6 GHz with a nVidia GeForce 7400 TC. The video acquisition is performed with usb Logitech fusion webcams connected to the computer via an usb hub. With a 320×240 resolution, 4 webcams streaming simultaneously provide 15 frames per second.

The computation time to create a new view is linearly dependent of the number of planes used, of the number of input images and of the resolution of the virtual view. The number of planes required depends on the scene. During our tests, we noticed that under 10 planes, the visual results became unsatisfactory and more than 60 planes did not improve the visual quality. Hence, in our experimentations, we used 60 planes to ensure an optimal visual quality. We set the virtual image resolution (output image) to 320×240. With such configuration, the number of input cameras is limited to 4 due to the GPU time computation. Our method reaches 15 frames per second. Figure 5 shows a real-view take exactly between two adjacent cameras, the corresponding created image and the difference. This difference is small enough to ensure good quality result.

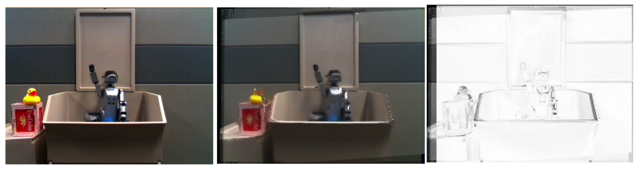

Figure 5. *Left* **: real view.** *Center* **: computed view. The virtual camera is placed between two adjacent input cameras .** *Right* **: difference between real and computed view.**

As shown in Figure 6, using 10 webcams provides a large range of available virtual view points. Thanks to our webcam management method. Hence, the range of available virtual view points can be extended without any visual quality loss. The user can prefer to decrease the base-line between the cameras. The Figure 7 shows two images created from 4 webcams. In the first case, the distance between two adjacent cameras is 10 cm, in the second case, 5 cm. We can see that reducing the base-line hugely improves the visual result.

In this system, the video acquisition is performed by the CPU and the new view computation by the GPU, hence these two process are independent.

In our tests, the bottle neck is the webcam acquisition frame-rate. This could be avoided by using other webcams, then the frame rate would be limited by the plane-sweep method, and especially by the virtual view resolution.

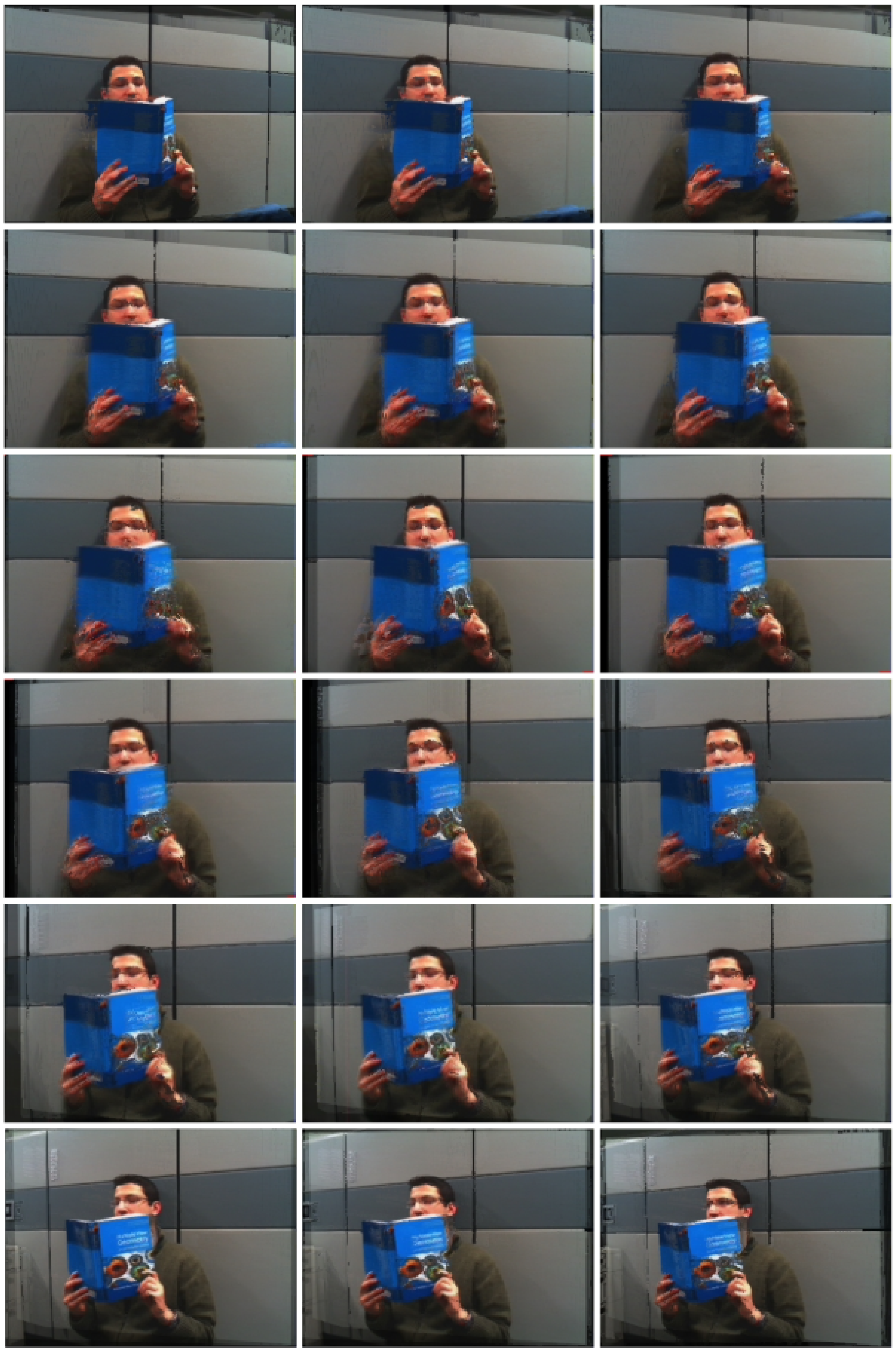

Figure 6. Each new view is computed on-line with a laptop using 4 cameras selected between 10. The scene was discretized with 60 planes and this method reaches 15 fps.

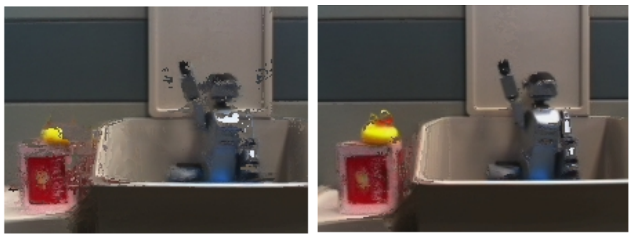

Figure 7. *Left* **: 10 cm base-line.** *Right* **: 5 cm base-line**

8 Conclusion

This paper presents an on-line Video-Based Rendering application using a plane-sweep algorithm that can be implemented on every consumer graphic hardware that supports fragment shaders. Our tests showed that this method combines low-cost hardware with high performances.

The proposed scoring method enhances the visual quality of the new views by using all input images together where other methods compute images by pair. We also present a video stream management that extends the number of potential webcams used to create a new view. This technique involves a better flexibility on the cameras' position and increases the visual result. Compared to others on-line VBR techniques, this method can handle the scene background, does not require more than one computer and provides high quality images.

As a future work, we intend to achieve an optimization on D_i planes repartition in order to increase the visual quality without adding any further planes.

Acknowledgment

This work has been supported by ”Foundation of Technology Supporting the Creation of Digital Media Contents” project (CREST, JST), Japan.

References

[1] R. T. Collins. A space-sweep approach to true multi-image. *In proc. Computer Vision and Pattern Recognition Conf.*, pages 358–363, 1996.

[2] J.-S. Franco and E. Boyer. Fusion of multi-view silhouette cues using a space occupancy grid. *In proc. of International Conference on Computer Vision ICCV'05*, pages 1747–1753, 2005.

[3] B. Goldlucke, M. A. Magnor, and B. Wilburn. Hardware accelerated dynamic light field rendering. *In proc. of Modeling and Visualization VMV 2002*, pages 455–462, 2002.

[4] S. D. R. I. Geys and L. Gool. The augmented auditorium : Fast interpolated and augmented view generation. *in proc, of European Conference on Visual Media Production, CVMP'05*, pages 92–101, 2005.

[5] T. Kanade, P. J. Narayanan, and P. Rander. Virtualized reality : concepts and early results. *In proc. of the IEEE Workshop on Representation of Visual Scenes*, page 69, 1995. In proc. of the IEEE Workshop on Representation of Visual Scenes.

[6] H.-P. S. M. Li, M. Magnor. Hardware-accelerated visual hull reconstruction and rendering. *In proc. of Graphics Interface (GI'03)*, pages 65–71, 2003.

[7] M. A. Magnor. *Video-Based Rendering*. ISBN : 1568812442, a k peters ltd edition, 2005.

[8] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, and L. McMillan. Image-based visual hulls. *in proc ACM SIGGRAPH 2000*, pages 369–374, 2000.

[9] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques For High-Performance Graphics And General-Purpose Computation*. ISBN-10: 0321335597, addison-wesley professional edition, 2005.

[10] B. Wilburn, N. Joshi, V. Vaish, E.-V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, and M. Levoy. High performance imaging using large camera arrays. *In proc. of ACM SIGGRAPH 2005*, 14:765–776, 2005.

[11] J. C. Yang, M. Everett, C. Buehler, and L. McMillan. A real-time distributed light field camera. *In proc. of the 13th Eurographics workshop on Rendering*, pages 77–86, 2002.

[12] R. Yang, G. Welch, and G. Bishop. Real-time consensus-based scene reconstruction using commodity graphics hardware. *In proc. of Pacific Graphics*, pages 225–234, 2002.

[13] Z. Zhang. A flexible new technique for camera calibration. *In proc. of IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:1330–1334, 2000.

[14] C. L. Zitnick, S. B. Kang, Matthew, and R. Szeliski. High-quality video view interpolation. *In proc. ACM SIGGRAPH 2004*, pages 600–608,, 2004.