HAL
open science

Real-time Free Viewpoint from Multiple moving Cameras

Vincent Nozick, Hideo Saito

To cite this version:

Vincent Nozick, Hideo Saito. Real-time Free Viewpoint from Multiple moving Cameras. Advanced Concepts for Intelligent Vision Systems (ACIVS), Aug 2007, Netherlands. pp.72-83. hal-00733823

HAL Id: hal-00733823

https://hal.science/hal-00733823

Submitted on 19 Sep 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Real-Time Free Viewpoint from Multiple Moving Cameras

Vincent Nozick[1,2] and Hideo Saito[2]

[1] Gaspard Monge Institute, UMR 8049, Marne-la-Vallée University, France
[2] Graduate School of Science and Technology, Keio University, Japan
{nozick,saito}@ozawa.ics.keio.ac.jp

Abstract. In recent years, some Video-Based Rendering methods have advanced from off-line rendering to on-line rendering. However very few of them can handle moving cameras while recording. Moving cameras enable to follow an actor in a scene, come closer to get more details or just adjust the framing of the cameras. In this paper, we propose a new Video-Based Rendering method that creates new views of the scene in live from four moving webcams. These cameras are calibrated in real-time using multiple markers. Our method fully uses both CPU and GPU and hence requires only one consumer grade computer.

1 Introduction

Video-Based Rendering (VBR) is an emerging research field that proposes methods to compute new views of a dynamic scene from video streams. VBR techniques are divided into two families. The first one, called *off-line* methods, focuses on the visual quality rather than on the computation time. These methods usually use a large amount of cameras or high definition devices and sophisticated algorithms that prevent them from live rendering. First, the video streams are recorded. Then the recorded data is computed off-line to extract 3d informations. Finally, the rendering step creates new views of the scene usually in real-time. This three-step approach (record - compute - render) provides high quality visual results but the computation time can be long compared to the length of the input video. The methods from the second family are called *on-line* methods. They are fast enough to extract information from the input videos, create and display a new view several times per second. The rendering is then not only real-time but also live.

Almost all the VBR techniques use calibrated input cameras and these calibrated cameras must remain static during the shot sequence. Hence it is impossible to follow a moving object with a camera. Using one or more moving cameras allows to come closer to an actor to get more details. This technique can also be used to adjust the framing of the cameras. Furthermore, if someone involuntary moves a camera, calibration update is not required. Hence this technique provides more flexibility in the device configuration.

In this article, we present a new live VBR method that handles moving cameras. For every new frame, each camera is calibrated using multiple markers laid on the scene. Contrary to most Augmented Reality applications, the multiple markers used in our method do not need to be aligned since their respective position are estimated during the calibration step. Our method uses four input webcams connected to a single computer. The lens distortion correction and the new views are computed on the GPU while the video stream acquisition and the calibration are performed by the CPU. This configuration fully exploits both CPU and GPU. Our method follows a plane-sweep approach and contrary to concurrent methods, a background extraction is not required. Therefore this method is not limited to render a unique object. Our method provides good quality new views using only one computer while concurrent methods usually need several computers. According to our knowledge, this is the first live VBR method that can handle moving cameras.

In the following parts, we propose a survey of previous works on both recent off-line and on-line Video-Based Rendering techniques. Then we explain the plane sweep algorithm and our contribution. Finally, we detail our implementation and we discuss experimental results.

2 Video-Based Rendering : Previous work

2.1 Off-lines Video-Based Rendering

The Virtualized Reality presented by Kanade et al. [1] is one of the first methods dealing with VBR. The proposed device first records the video streams from 51 cameras and then computes frame by frame a depth map and a reconstruction for every input camera. Finally, the new views are created using the reconstruction computed from the most appropriate cameras. Considering the amount of data to compute, the depth map and reconstruction process can take very long. Therefore this method is hardly compatible with live rendering.

Goldlucke et al. [2] follows the same off-line approach using 100 cameras. Zitnick et al. [3] also uses this approach but with around ten high definition cameras. The depth maps are computed using a powerful but time-consuming segmentation method and the rendering is performed with a layered-image representation. This method finally provides high quality new views in real-time. The Stanford Camera Array presented by Wilburn et al. [4] uses an optical flow approach and provides real-time rendering from 100 cameras. Franco and Boyer [5] provide new views from 6 cameras with a Visual Hulls method.

VBR methods designed to handle moving cameras are very rare. Jarusirisawad and Saito [6] propose a projective grid space method that can use uncalibrated cameras. This method does not use marker but the cameras' movements are limited to pure rotation and zooming. This method also need a significant

amount of time to compute a new view and thus is not well suited to real-time rendering.

All these VBR methods use a three-step approach (record - compute - render) to create new views. Naturally, they can take advantage of the most powerful algorithms even if they are time consuming. Furthermore, since most of these methods use a large amount of information, the computing process becomes very long, but the visual result is usually excellent.

2.2 Live Video-Based Rendering

Contrary to off-line methods, live (or on-line) methods are fast enough to extract informations from the input videos, create and display a new view several times per second. However powerful algorithms such as global optimization are not suited for real-time implementation, and thus we can not expect equivalent accuracy and visual results from both off-line and on-line methods. Furthermore, only a few VBR methods reach on-line rendering and, according to our knowledge, none of them handles moving cameras. And since live rendering imposes severe constraints on the choice of the algorithms used, it becomes difficult to adapt a live method for moving cameras.

Currently, the Visual Hulls algorithm is the most popular live VBR method. This method first computes a background extraction on every frame such that it remains only the main "object" of the scene on the input images. The 3d shape of this object is then approximated by the intersection of the projected silhouettes. Several on-line implementations have been proposed and most of them are described in [7]. The most significant method is probably the Image-Based Visual Hulls proposed by Matusik et al. [8]. This method reaches real-time and live rendering using four cameras connected to a five computer cluster. The easiest method to implement is very likely the Hardware-Accelerated Visual Hulls presented by Li et al. [9]. The main drawback of these methods is the impossibility to handle the background of the scene since only one main "object" can be rendered. Furthermore, the Visual Hulls methods usually require several computers, which makes their use more difficult. On the other hand, these methods have the ability to place the input cameras far from each other, for example around the main object. However this advantage becomes a big constraint for real-time calibration since few cameras could see common calibration markers.

Yang et al. [10] propose a distributed Light Field using a 64-camera device based on a client-server scheme. The cameras are controlled by several computers connected to a main server. Only those image fragments needed to compute the new view are transferred to the server. This method provides live rendering but requires at least height computers for 64 cameras and additional hardware. Technically, this method can probably be adapted to moving cameras but it may become difficult to move correctly 64 cameras .

Finally, some plane-sweep methods reach on-line rendering. Yang et al. [12] compute new views in live from five cameras using four computers. Geys et al. [13] combine a plane sweep algorithm with a 3d shape optimization method and provide live rendering from three cameras and one computer. Since our method belongs to the latter family, we will expose the basic plane-sweep algorithm and [12, 13] contribution in the next section. Then we will detail our rendering method and the camera calibration step.

3 Plane-Sweep algorithm

3.1 Overview

Given a small set of calibrated images from video cameras, we wish to generate a new view of the scene from a new viewpoint. Considering a scene where objects are exclusively diffuse, we first place the virtual camera cam_x and define a $near$ plane and a far plane such that every object of the scene lies between these two planes. Then, we divide space between $near$ and far planes in parallel planes D_i in front of cam_x as shown in Fig. 1.

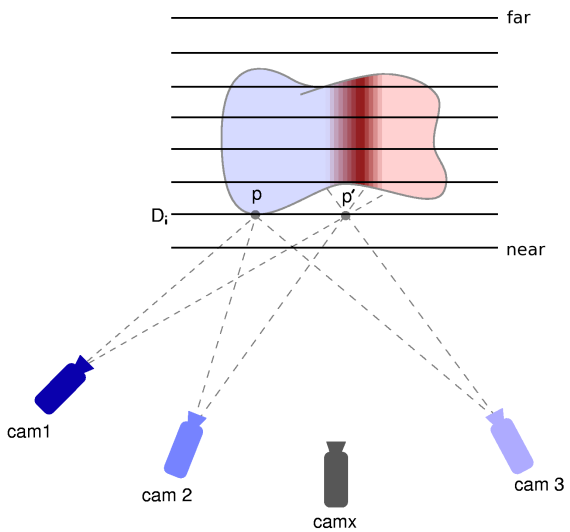

Fig. 1. Plane-sweep : geometric configuration

Let's consider a visible object of the scene lying on one of these planes D_i at a point p. Then this point will be seen by every input camera with the same color (i.e. the object color). Consider now a point p' that lies on a plane but not on the surface of a visible object. As illustrated on Fig. 1, this point will probably not

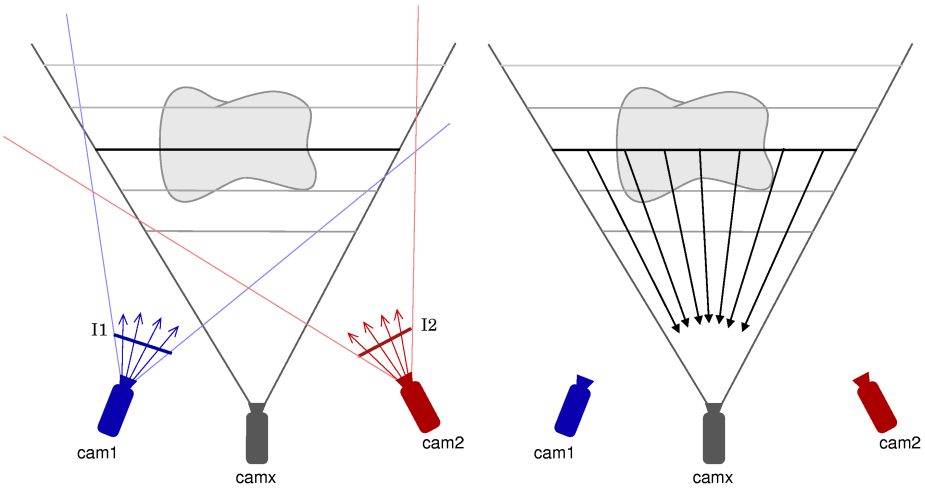

Fig. 2. *Left* : all input images are projected on the current plane. A score and a color are computed for every point of this plane. *Right* : these computed scores and colors are projected on the virtual camera.

be seen by the input cameras with the same color. Therefore, points on planes D_i whose projection on every input camera provides a similar color potentially correspond to the surface of an object of the scene.

A usual way to create a new image is to process the planes D_i in a back to front order. For each pixel p of each plane D_i, a score and a color are computed according to the matching of the projected colors (Figure 2). When every pixel p of a plane is computed, every score and color are projected on the virtual camera cam_x. The final image is computed in a z-buffer style : consider a point p projected on a pixel of the virtual image. This pixel's color will be updated only if the score of p is better that the current score. We note that, thanks to this plane approach, this method is well suited for use on graphic hardware.

3.2 Previous implementation

The plane sweep algorithm was first introduced by Collins [11]. Yang et al. [12] propose an real-time implementation using register combiners. They first place the virtual camera cam_x and choose among the input cameras a reference camera that is closest to cam_x. For each plane D_i, they project the input images, including the reference image. During the scoring stage, they compute for every pixel p of D_i a score by adding the Sum of Squared Difference (SSD) between each projected image and the projected reference image. This method provides real-time and on-line rendering using five cameras and four computers, however the input cameras have to be close to each other and the navigation of the vir-

tual camera should lie between the viewpoints of the input cameras, otherwise the reference camera may not be representative of cam_x. Lastly, there may appear discontinuities in the computed video when the virtual camera moves and changes its reference camera.

Geys et al. [13] combined Yang et al. plane-sweep implementation with an energy minimization method based on a graph cut algorithm to create a 3d triangle mesh. This method provides real-time and on-line rendering using 3 cameras and only one computer. However this method requires a background extraction and only compute a 3d mesh for non-background objects.

4 Our scoring method

The score computation is a crucial step in the plane sweep algorithm. Both visual results and speedy computation depend on it. Previous methods computes scores by comparing input images with **a/the** reference image. Our method aims to avoid the use of such reference image that is usually not representative of the virtual view. We also try to use every input image together rather than to compute images by pair. However, since the scoring stage is performed by the graphic hardware, only simple instructions are supported.

An appropriate solution is then to use variance and average tools. Consider a point p lying on a plane D_i. The projection of p on each input image j provides a color c_j. We propose to set the score as the variance computed from every c_j and the final color as the average color of the c_j. If every input colors c_j match together, this method will provide a small variance which corresponds to a high score. Furthermore, the average color will be highly representative of the c_j. If the input colors c_j mismatch, the provided score will be low since the computed variance will be high. In the latter case, the average color will not be representative of the input colors c_j but since the score is low, this color will very likely not be selected for the virtual image computation. Finally, our plane-sweep implementation can be explained 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 jth input image
    - → compute the average and the variance of $\{c_j\}_{j=1...n}$
    - → set the color and the score of p to the computed average and variance
  - project 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 previous ones
      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 handles dynamic backgrounds.

5 On-line calibration from multiple markers

Like most VBR techniques, our method requires calibrated cameras. Furthermore, since the cameras are allowed to move, the calibration parameters must be updated for every frame. The calibration should be accurate but also real-time for multiple cameras. To satisfy this constraints, we opted for a marker-based approach. The markers we used are 2D patterns drawn in black squares. They are detected and identified by *ARtoolkit* [14], a very popular tool for simple on-line Augmented Reality applications. Using only one marker is usually not enough to calibrate a camera efficiently. Indeed, the marker detection may not be accurate or the marker may not be detected (detection failure or occlusion). Multiple markers reduce the detection failure problem and provide better results.

In most of the multiple markers applications, the markers are aligned and their respective position must be known. To decrease the constraints on the markers layout, some methods like [15] or [16] use multiple markers with arbitrary 3d positions. In our case, the cameras view-point can change every time, then it seems to be easier to increase the number of markers seen by a camera if they are close to each other. Thus a coplanar layout is well suited for our VBR method. In the following part, we present a method using multiple markers with arbitrary position and size. In this method, *ARtoolkit* is used only to provide markers' position in the image coordinates, but not for calibration.

First, the cameras internal parameters, should be preliminary computed. Then, the full calibration part can begin. The user sets some markers in a planar configuration. They can have different sizes and any layout is satisfactory. A reference marker should be chosen to be the origin of the scene referential. Then one of the input camera takes a picture containing all the markers so the geometrical relationship between the markers could be estimated. Indeed, a homography H between this picture and the reference marker is computed (see figure 3). Applying H on the pixel coordinate of every detected marker will provide its position in the scene referential.

Then, every moving camera can be calibrated in real-time. At least one marker should appear in an image to compute a calibration matrix. First, every detected marker is computed independently. A projection matrix is estimated by Zhang method [17] using correspondences between the marker pixel coordinates and its position in the scene referential previously computed. Then the final projection matrix is set as the average projection matrix computed from every marker.

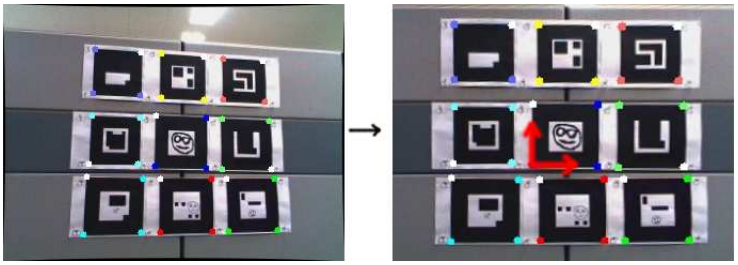

Fig. 3. Estimation of the relationship between every marker by homography.

Thus in this method, both rotation and translation are handled. To make the use of our VBR method easy, we propose to define the *far* plane as the plane containing the markers.

6 Implementation

We implemented this method in C++ with OpenGL. The video streams acquisition is performed using Video for Linux. During the calibration step, we use *ARtoolkit* [14] only for the marker detection. The full calibration is then computed as explained in section 5. The plane that contains all the markers is assimilated to the *far plane*. Thus the user just have to define the depth of the scene. Nevertheless, these two planes can also be set automatically using a precise stereo method as described in [13]. Naturally, the calibration parameters can be locked and unlocked during the shooting. Thus, even if all the markers are occluded, the camera calibration remains correct, but in that case, the cameras have to remain static during this time interval.

Our method is specially well-suited to be used with webcams. However this kind of cameras are usually subject to lens distortion. Since the markers used for the calibration step can appear every where in the input images and not only in the central part, lens distortion correction is indispensable. In our method, we only focused on the radial distortion correction [18]. Our experiments shows that this correction can not be done by CPU in real-time. Indeed, the CPU is already fully exploited by the video stream acquisition, the markers detection, the camera calibration and others tasks related to the plane sweep algorithm. Hence the correction is performed by the GPU using fragment shaders. This step is done off-screen in one pass for each input image using Frame Buffer Objects. Implementation indications can be found on [19]. Skipping the radial distortion correction will have repercussion on both the calibration accuracy and the score computation. Then the visual result will be a bit decreased.

Concerning the new view computation, the user should define the number k of planes D_i used for the rendering. The new view computation requires k passes.

Fig. 4. Images computed in live from four cameras.

Each plane D_i is drawn as multi-textured `GL_QUADS`. Multi-texturing provides an access to every texture simultaneously during the scoring stage. The scores are computed thanks to fragment shaders using the algorithm discribed 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 summarize, the CPU performs the video stream acquisition, the camera calibration and the virtual camera control. The GPU corrects the lens distortion and creates the new view. This configuration fully uses the capability of both CPU and GPU. We tried other configurations but the result was not real-time.

7 Results

We tested our method with an Intel Core2 1.86 GHz with a GeForce 7900 GTX. The video acquisition is performed by four usb Logitech fusion webcams and reaches 15 frames per second with a 320×240 resolution.

The computation time to create a new view is linearly dependent on the number of planes used, on the number of input images, and on the resolution of the

virtual view. The number of planes required depends on the scene. In our tests, the experimentations showed that under 10 planes, the visual result became unsatisfying and over 60 planes, the visual results are not improved. Since the bottleneck of our method is the video stream acquisition, we used 60 planes in our experiments. Finally, we set the virtual image resolution to 320×240. With this configuration, our method reaches 15 frames per second.

A normal base-line configuration for our system is a roughly 30° angle between the extreme cameras and the center of the scene. The cameras do not have to be aligned. Figure 4 shows nine views corresponding to an interpolation between the four cameras. Figure 5 depicts a virtual view taken at mid-distance between two adjacent cameras with the same camera configuration used in Figure 4. The middle image corresponds to the real image and the right image is the difference between the virtual and the real image. Despite some mismatching in objects' borders, our method provides good accuracy.

Whatever the markers layout are, our real-time calibration method provides accurate calibration for our plane sweep method as long as every input camera can detect at least one or two markers. Thanks to this method, the cameras can move in the scene without any repercussion on the visual result. Figure 6 shows two images created during the same sequence where cameras have been moved. The markers can have different size but all of them should be detected and identified in a single image during the first calibration step.

In our tests, the bottle neck of the method is the webcam acquisition frame-rate but some others webcams provides higher frame rates. Our application speed would then be limited by the plane-sweep method, and especially by the virtual view resolution.

Fig. 5. *Left* : new view created at mid-distance between two adjacent cameras, *Middle* : real image, *Right* : difference between the real image and the computed view.

Currently, four webcams is our upper limit for real-time new view computation. Using more cameras would increase the visual result quality and more powerful GPU would probably help to increase the number of cameras, but the real-time video-stream acquisition would become a problem. Our experiments also show that using only three cameras slightly decreases the visual result and restricts the cameras configuration to smaller base-lines.

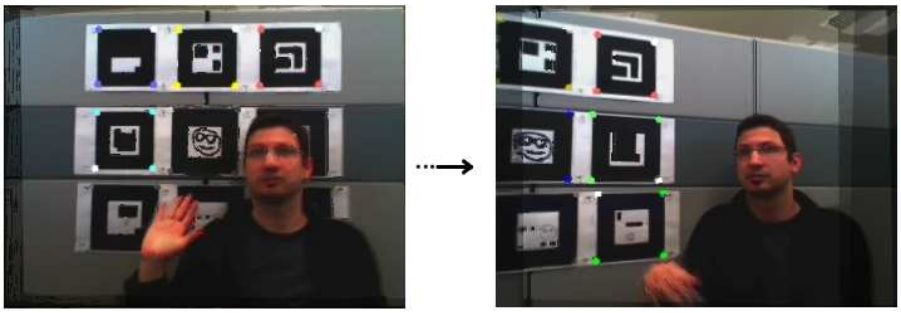

Fig. 6. During the same video sequence, the input cameras can be moved.

8 Conclusion

In this article we present a live Video-Based Rendering method that handles moving cameras and requires only a consumer grade computer. The new view is computed from four input images and our method follows a plane sweep approach. Both the CPU and the GPU are fully exploited. The input cameras are calibrated using multiple markers. These markers must be coplanar but their disposition and their size do not have to be known in advance such the user can choose the most adequate configuration. Our method reaches live rendering with four webcams. These cameras can be moved to follow an actor or to focus on a specific part of the scene.

Our implementation shows that it is possible to combine live Video-Based Rendering with moving cameras using markers. Concerning future works, we intend to enhance our method using real-time calibration without markers.

References

1. Takeo Kanade, P. J. Narayanan and Peter Rander: Virtualized reality : concepts and early results, in proc. of the IEEE Workshop on Representation of Visual Scenes, page: 69, 1995.
2. Bastian Goldlucke, Marcus A. Magnor, Bennett Wilburn: Hardware accelerated Dynamic Light Field Rendering, in proc. of Modelling and Visualization VMV 2002, aka Berlin, Germany, pages 455-462, November 2002.

3. C. Lawrence Zitnick, Sing Bing Kang, Matthew and Richard Szeliski: High-quality video view interpolation, in proc. ACM SIGGRAPH 2004, pages 600-608, august 2004.
4. Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz and Marc Levoy : High Performance Imaging Using Large Camera Arrays, in proc. of ACM SIGGRAPH 2005, pages 765-776, July 2005.
5. Jean-Sbastien Franco and Edmond Boyer, Fusion of Multi-View Silhouette Cues Using a Space Occupancy Grid, in proc. of International Conference on Computer Vision ICCV'05, page 1747-1753, 2005.
6. Songkran Jarusirisawad and Hideo Saito, New Viewpoint Video Synthesis in Natural Scene Using Uncalibrated Multiple Moving Cameras, in International Workshop on Advanced Imaging Techniques IWAIT 2007, Pages 78-83, 2007.
7. Marcus A. Magnor : Video-Based Rendering. Editor: A K Peters Ltd, ISBN : 1568812442, 2005.
8. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler and Leonard McMillan : Image-Based Visual Hulls, in proc ACM SIGGRAPH 2000, pages 369-374, 2000.
9. Ming Li, Marcus Magnor, Hans-Peter Seidel : Hardware-Accelerated Visual Hull Reconstruction and Rendering, in proc. of Graphics Interface GI'03, pages 65-71, Halifax, Canada, June 2003.
10. Jason C. Yang, Matthew Everett, Chris Buehler and Leonard McMillan : A real-time distributed light field camera, in proc. of the 13th Eurographics workshop on Rendering, pages 77-86, Italy, 2002.
11. Robert T. Collins : A Space-Sweep Approach to True Multi-Image, in proc. Computer Vision and Pattern Recognition Conf., pages 358-363, 1996.
12. Ruigang Yang, Greg Welch and Gary Bishop : Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware, in proc. of Pacific Graphics, pages 225-234, 2002.
13. Indra Geys, Stefaan De Roeck and Luc Van Gool : The Augmented Auditorium : Fast Interpolated and Augmented View Generation, in proc, of European Conference on Visual Media Production, CVMP'05, page 92-101, 2005.
14. M. Billinghurst, S. Campbell, W. Chinthammit, D. Hendrickson, I. Poupyrev, K. Takahashi, and H. Kato, Magic book: Exploring transitions in collaborative ar interfaces, in proc. of SIGGRAPH 2000, page 87, 2000.
15. Yuko Uematsu and Hideo Saito : AR registration by merging multiple planar markers at arbitrary positions and poses via projective space, in proc. of ICAT2005, pages 4855, 2005.
16. Jong-Hyun Yoon, Jong-Seung Park and Chungkyue Kim : Increasing Camera Pose Estimation Accuracy Using Multiple Markers, in proc. of 16th International Conference on Artificial Reality and Telexitsence, ICAT 2006, pages 239-248, 2006
17. Zhengyou Zhang : A flexible new technique for camera calibration, in proc. of IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, pages 1330-1334, 2000.
18. Richard Hartley and Andrew Zisserman, Multiple View Geometry in Computer Vision, Second Edition,Cambridge University Press, ISBN: 0521540518, 2004.
19. Matt Pharr, Randima Fernando, GPU Gems 2: Programming Techniques For High-Performance Graphics And General-Purpose Computation, Addison-Wesley Professional, ISBN-10: 0321335597, 2005.