Plane-Sweep Algorithm: Various Tools for Computer Vision

Vincent Nozick
Graduate School of Science and Technology,
Keio University, Japan
nozick@ozawa.ics.keio.ac.jp

François de Sorbier
Université Paris-Est, France
LabInfo IGM
UMR CNRS 8049
fdesorbi@univmlv.fr

Hideo Saito
Graduate School of Science and Technology,
Keio University, Japan
saito@ozawa.ics.keio.ac.jp

Abstract

Recent research on computer vision has made significant progress in 3d reconstruction and free-viewpoint video however most of these methods are not suited for real-time rendering. This paper presents a Video-Based Rendering method that provides online new viewpoints of the scene from a set of webcams. Our method follows a plane-sweep approach perfectly suited for GPU implementation. This paper mainly focuses on different implementations of this method for different purposes. Indeed, our basic plane-sweep technique uses 4 input cameras to create online new views. However this method can be modified to be used with up to 10 cameras or more by a camera selection process. This method can also be adapted to handle moving input cameras using a real-time camera calibration technique. Moreover, this method can easily render a depth map instead of a new view. Finally, we explain how the plane-sweep algorithm can be modified to create multiple new views simultaneously. This latter application is mainly designed for autostereoscopic displays application. Implementation and performance are detailed for all these plane-sweep methods.

Keywords

Video-based rendering, GPU, real-time rendering, on-line rendering.

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 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. 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.

In this article, we present a VBR method that creates new views of the scene on-line. This method is especially well suited to be used with GPU. Hence, we detail some real-time computer vision applications optimized to fully use both CPU and GPU. In the following parts, we propose a survey of previous works on recent on-line Video-Based Rendering techniques. The next part explains the plane-sweep algorithm and our contributions. Then, we present some real-time applications and their implementation.

2 Online 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 probably 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 [1]. The most accurate on-line Visual Hulls method seems to be the Image-Based Visual Hulls presented by Matusik et al. [2]. This method creates news views in real-time from four cameras. Each camera is controlled by one computer and an additional computer creates the new views. The method proposed by Li et al. [3] 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. [4]. 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). Since our method belongs to the latter family, we will expose the basic plane-sweep algorithm in the next section. Then we will detail our contribution.

3 Plane-Sweep Algorithm

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 on 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 matching of the colors found. A good score corresponds to similar colors. This process is illustrated on Figure 2.

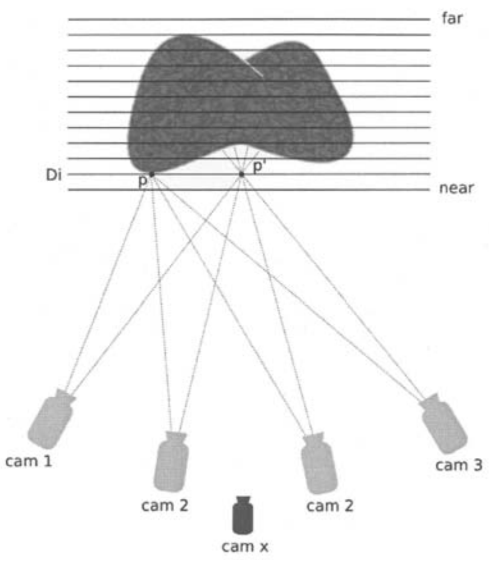

Figure 1. Plane-sweep: geometric configuration.

Then, the computed scores and colors are projected on the virtual camera camx. 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 Di+1 is computed. The final image is obtained when every plane is computed.

The plane-sweep algorithm introduced by Collins [5] was adapted to on-line rendering by Yang et al. [6] using register combiners. The system chooses a reference camera that is closest to camx. During the process of a plane Di, 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 five cameras and four 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 camx. 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 [7] 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.

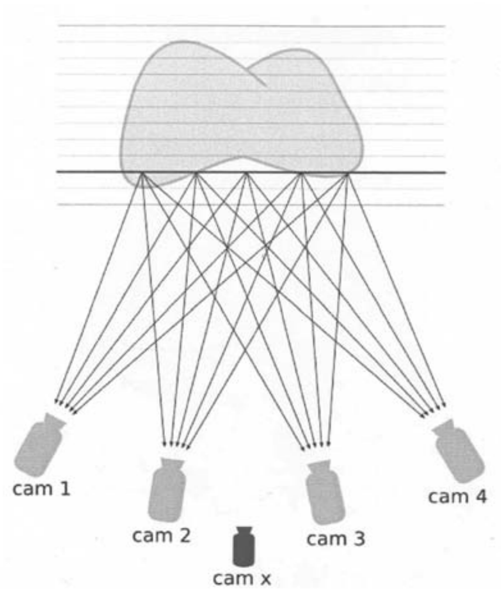

Figure 2. Every point of the current plane is projected on the input images. A score and a color are computed for these points according to the matching of the colors found.

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 three cameras and only one computer. However, the background geometry must be static.

4 Proposed Method

4.1 Score Computation

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.

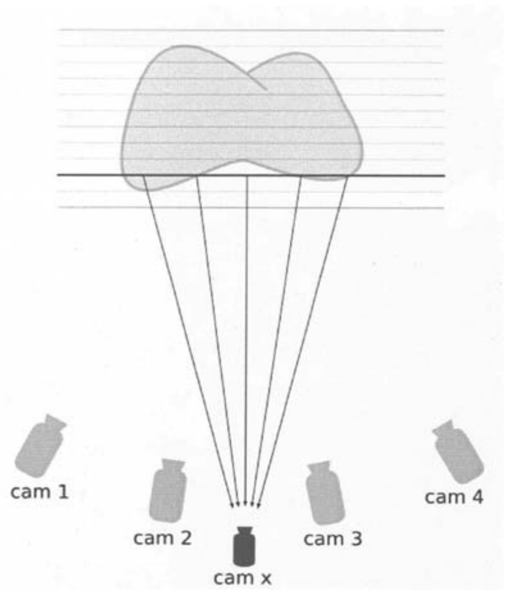

Figure 3. The computed scores and colors are projected on the virtual camera.

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 $p$ :
          $color_p = \text{average}(c_j)_{j=1...n}$
        - compute the score of $p$ :
          $score_p = \text{sum}((c_j - \text{color})^2)_{j=1...n}$
    • 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
```

4.2 Implementation and Results

Our implementation is designed to be used with four webcams, connected to a single computer. The camera calibration is performed using Zhang method [8]. 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. [7]. 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].

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. Multi-texturing provides an access to every texture simultaneously during the scoring stage. Then, this fragment program computes each score and color using the algorithm before-mentioned. 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.

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, four 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 4 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.

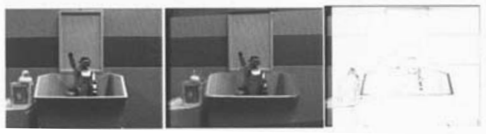

Figure 4. ***Left*****: real view.** ***Center*****: computed view. The virtual camera is placed between two adjacent input cameras.** ***Right*****: difference between real and computed view.**

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 5).

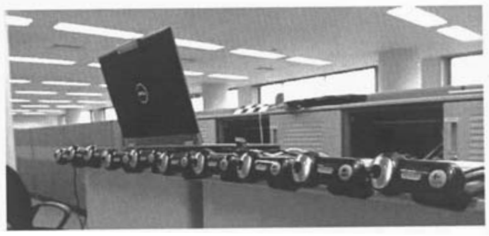

Figure 5. Ten webcams connected to a laptop via a USB hub.

Considering the position of the virtual camera, the system selects the four 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 four 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. Figure 6 shows 18 new views computed in real-time from ten cameras.

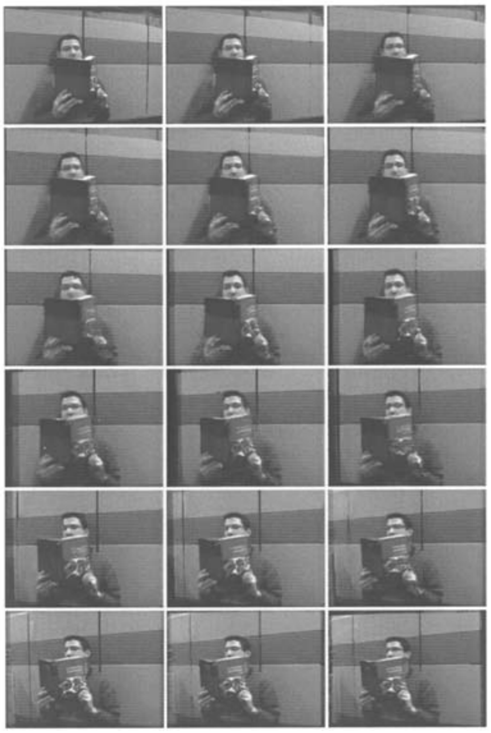
Figure 6. Each new view is computed on-line with a laptop using 4 cameras selected between 10. The scene was discretized with 60 planes and this method reaches 15 fps.

6 Depth-Map Rendering

Real-time depth map estimation is a challenging topic where the planes-sweep method provides promising results. Woetzel et al. [10] propose a plane-sweep technique to compute real-time depth maps using an optimization of Yang et al. [6] approach. Similarly, we propose an adaptation of our method to generate real-time depth maps. This adaptation requires very few changes in our initial plane-sweep program. For a pixel candidate p lying on a plane D_i, we just have to provide the depth of D_i instead of the color of p. Hence, in our implementation, to compute a depth-map rather than a new view just involves to set the `gl_FragColor` to the `gl_FragCoord.z` value in the fragment shader program. To enhance the depth map accuracy, we added a real-time background substraction on the input video stream. Figure 7 illustrates some real-time results from four webcams.

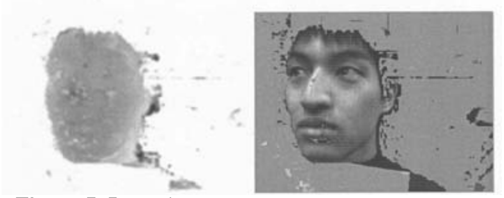
Figure 7. Real-time depth map computation from four webcams.

7 Moving Cameras

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 coming 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 moving cameras device provides more flexibility in the device configuration.

We propose a real-time calibration method designed to be used with our plane-sweep method. The calibration is accurate but also real-time for multiple cameras. Our method follows a marker-based approach: the 2D markers are detected and identified by ARtoolkit [11], a very popular tool for simple on-line Augmented Reality applications.

Using only one marker is usually not enough to calibrate a camera efficiently. 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 [12] or [13] use multiple markers with arbitrary 3D positions. In our case, the cameras view-point can change every time, hence 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 our method, ARtoolkit is used only to provide markers' position in the image coordinates, but not for calibration. The user sets some markers in a planar configuration. They can have arbitrary position and size. A reference marker should be chosen to be the origin of the scene referential. Then one of the input cameras 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 (Figure 8).

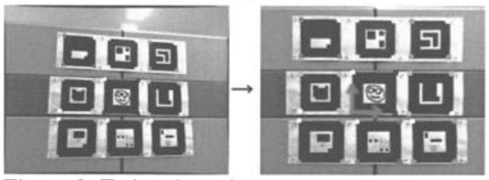
Figure 8. Estimation of the relationship between every marker by homography.

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 in the same referential. At least one marker should appear in an image to compute a calibration matrix. Every detected marker is computed independently. A projection matrix is estimated by Zhang method [8]. Then the final projection matrix is set as the average projection matrix computed from every marker. Thus in this method, both rotation and translation are handled. This method reaches real-time rendering thanks a combination of the GPU and the CPU. Figure 9 depicts some real-time results.

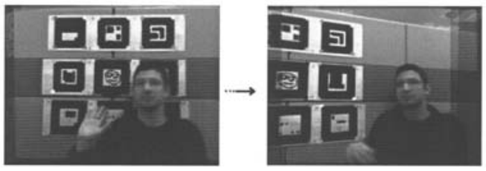
Figure 9. During the same video sequence, the input cameras can be moved.

8 Multiple-View Rendering

In this section, we explain how our VBR method can be adapted to multiple view rendering, i.e. how our method can be modified to render simultaneously several new views of the scene from different viewpoint in real-time.

A major application of multiple view rendering is to provide a set of stereo images for autostereoscopic displays. Indeed, autostereoscopic devices require several images of the same scene from different viewpoints. Using ten or more cameras can provide enough views for an autostereoscopic display but even with a low resolution, real-time video stream acquisition is a serious issue with a single computer. VBR methods are a good alternative to this approach since they can provide new views of the scene from a restricted set of videos and thus decrease the number of required cameras.

As presented in section 2, few VBR methods reach on-line rendering. Moreover, autostereoscopic display applications require not only on-line VBR methods but also methods that can create simultaneously several new views of the scene for every frame.

The visual hulls methods is suited for an "all around" camera configuration but not for a dense aligned camera configuration required for autostereoscopic display applications. The distributed Light Field proposed by Yang et al. [4] requires at least 8 computers for 64 cameras and additional hardware. Thus this method is incompatible with a commercial use of stereoscopic applications.

Actually, most of on-line VBR methods already fully use the available computer capability to reach real-time rendering, thus we can hardly expect real-time rendering for multiple views without any optimization. The plane-sweep algorithm is well suited for such optimization thanks to the space decomposition using planes. Indeed, scores and colors computed on every plane represent local information of the scene. This score and color computation, which are a central task in the plane-sweep algorithm, can be shared among every virtual view and hence provide a consequent gain of computation time.

The plane-sweep method can be modified in a k+1 passes algorithm, where k is the number of virtual cameras, to provide on-line multiple new views. For every plane D_i (Figure 1), the score and color of every point is computed in a first pass. This pass is absolutely independent of the number of virtual views to create. The information computed during this pass is then projected on every virtual view in k passes. During these last k passes, color and score information is updated on every successive virtual camera. The k+1 passes are repeated until every plane D_i is computed. Hence our previous method can be modified as follows:

```
● reset the scores and colors of the virtual cameras $V_{j(1...k)}$
● **for** each plane $D_i$ from *far* to *near*
  • **for** each point (fragment) $p$ of $D_i$
    - compute a score *score* and a color *color*.
    - store the results *color* and *score* in an array :
        $T(p) = (color, score)$
  • **for** each camera $cam_j$
```

```
    - for each point (fragment) $p$ of $D_i$
        ◆ find the projection $q_{j,p}$ of $p$ on $cam_j$.
          $V_j(q_{j,p})$ contains previous color and score information on $cam_j$ at the position $q_{j,p}$
        ◆ if the score on $T(p)$ is better than the score stored on $V_j(q_{j,p})$
          then $V_j(q_{j,p}) = T(p)$
• convert each $V_j$ into images
```

Concerning the implementation, the use of the z-test for the multiple view method would imply that every new view is rendered on the screen. Thus the screen resolution would limit the number of new view that can be computed. We propose a method where every process is done off-screen using Frame Buffer Object. RGBA textures are assigned to every virtual view and an additional texture is used for the color and score computation. The color is stored in the RGB component and the score in the alpha component. The virtual camera's texture will replace the frame buffer used on the single view method.

The score and color computation of a plane does not differ from the single view method except that the rendering is performed on a texture. Naturally the rendering has to be associated to a projection matrix. We select the central virtual camera as a reference camera for this projection. Then, every virtual camera involves an additional rendering pass. During a pass, the score and color texture is projected on the current plane using the reference camera projection matrix. The textured plane is then projected on the virtual camera using fragment shaders. The texture associated to the current virtual camera is used for both rendering and reading the last selected scores and colors. The fragment program decides to update fragment information or to keep the current texture value according to the last selected scores as described in the before-mentioned algorithm. After the last plane computation, the virtual camera's texture can be extracted to provide the new images of the virtual views.

The number of virtual views depends on the application. In our case, we tested our system with 6, 9, 12, 15 and 18 virtual cameras set between adjacent input cameras. The speed results obtained with such configuration are shown on Table 1. Our test includes compression and transfer of both virtual views and input images. Table 1 also includes the frame rate of the classic method witch computes independently every virtual view. Our tests indicate that our method provides especially good results for a large number of virtual views. Compared to the classic method, our method is at least more than twice faster for 6 virtual views and is four time faster for 18 virtual views without any loss of quality.

Number of virtual views	Number of total views	Frame rate (fps)	Classic metho d (fps)
6	10	11.2	3.8
9	13	10	2.9
12	16	8.7	2.4
15	19	7.6	1.9
18	22	7	1.6

Table 1 - frame rate and number of virtual views.

Figure 10 depicts a sample result for a 12 virtual views configuration. Input images are displayed on the diagonal. The visual quality of a virtual view varies with its distance from input cameras and decreases for a virtual view located exactly between two input cameras. However, autostereoscopic display provides two views per user (right and left eyes) and the fusion of the two images decreases the imperfection impact. As shown on Figure 10, stereoscopic pairs (parallel-eyed viewing) are very comfortable. In addition, the base-line between the extreme right and left views are perfectly suited to autostereoscopic display application. More implementation details are available on Nozick and Saito [14].

9 Conclusion

This paper presents several on-line Video-Based Rendering applications 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. We propose an effective 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. We also show how our method can be modified to provide real-time depth maps rather than new views of the scene. Then we present a combination of the plane-sweep algorithm with a real-time calibration method to handle moving cameras. Finally, we present an adaptation of our method to provide simultaneously multiple views of a scene from a small set of webcams. Our multiple view method shares the 3D data computation for every virtual view and speeds up the computation time more than four times compared to the single view method for the same number of new views. The rendering is on-line and provides high quality stereoscopic views. This method is especially designed for autostereoscopic display. According to our knowledge, there does not exist other VBR method that provides equivalent results with such configuration.

Acknowledgments

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

References

[1] Marcus A. Magnor, Video-Based Rendering, A K Peters Ltd, ISBN : 1568812442, 2005.
[2] Wojciech Matusik , Chris Buehler, Ramesh Raskar, Steven J. Gortler and Leonard McMillan, Image-Based Visual Hulls, in proc. of ACM SIGGRAPH 2000, pp. 369-374, 2000.
[3] M. Li, Magnor and H.-P. Seidel, Hardware-Accelerated Visual Hull Reconstruction and Rendering, in proc. of Graphics Interface (GI'03), pp. 65-71, 2003.
[4] 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, pp. 77-86, 2002.
[5] Robert T. Collins, A Space-Sweep Approach to True Multi-Image, in proc. of Computer Vision and Pattern Recognition Conf., pp. 358-363, 1996.
[6] Ruigang Yang , Greg Welch and Gary Bishop, Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware, in proc. of Pacific Graphics, pp. 225-234, 2002.
[7] I. Geys, S. De Roeck and L.Van Gool, The Augmented Auditorium: Fast Interpolated and Augmented View Generation, in proc. of European Conference on Visual Media Production, CVMP'05, pp. 92-101, 2005.
[8] Z. Zhang, A flexible new technique for camera calibration, in proc. of IEEE Transactions on Pattern Analysis and Machine Intelligence, volume 22, pp. 1330-1334, 2000.
[9] Matt Pharr and Randima Fernando, GPU Gems 2: Programming Techniques For High-Performance Graphics And General-Purpose Computation, Addison-Wesley Professional, ISBN-10: 0321335597, 2005.

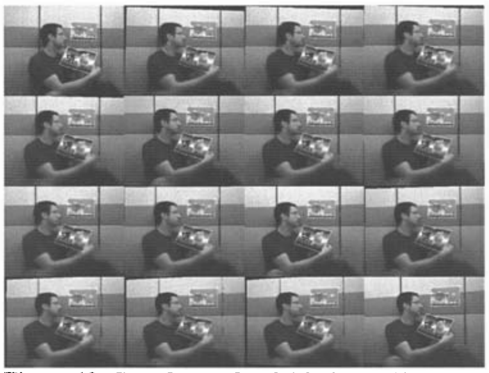
Figure 10. Sample result of 16 views: 12 virtual views and 4 input images on the diagonal. Parallel eyed viewing provides stereoscopic images.

[10] Woetzel, Jan, Koch and Reinhard, Multicamera real-time depth estimation with discontinuity handling on PC graphics hardware, in proc. of 17th International Conference on Pattern Recognition (ICPR 2004), pp. 741-744, 2004.
[11]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, pp. 87, 2000.
[12]Yuko Uematsu and Hideo Saito: AR registration by merging multiple planar markers at arbitrary positions and poses via projective space, in proc. of ICAT2005, pp. 4855, 2005.
[13] 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, pp. 239-248, 2006
[14] Vincent Nozick and Hideo Saito, *Multiple View Computation for Multi-Stereoscopic Display*, in proc. of 2007 IEEE Pacific-Rim Symposium on image and video Technology (PSIVT 2007), Vol. 4872, pp. 399-412, Santiago, Chile, December 17-19, 2007.