International Journal of Automation and Computing 05(3), July 2008, 257-267
DOI: 10.1007/s11633-008-0257-y

# On-line Free-viewpoint Video: From Single to Multiple View Rendering

Vincent Nozick* Hideo Saito

Graduate School of Science and Technology, Keio University, Kohoku-ku 223-8522, Japan

**Abstract:** In recent years, many image-based rendering techniques have advanced from static to dynamic scenes and thus become video-based rendering (VBR) methods. But actually, only a few of them can render new views on-line. We present a new VBR system that creates new views of a live dynamic scene. This system provides high quality images and does not require any background subtraction. Our method follows a plane-sweep approach and reaches real-time rendering using consumer graphic hardware, graphics processing unit (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. We also present an adaptation of our plane-sweep method to create simultaneously multiple views of the scene in real-time. Our system is especially designed for stereovision using autostereoscopic displays. The new views are computed from 4 webcams connected to a computer and are compressed in order to be transfered to a mobile phone. Using GPU programming, our method provides up to 16 images of the scene in real-time. The use of both GPU and CPU makes this method work on only one consumer grade computer.

**Keywords:** Video-based rendering (VBR), free-viewpoint video, view interpolation, graphics processing unit (GPU), webcam, stereovision, autostereoscopic.

## 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 focus on the visual quality rather than on the computation time. They use a large amount of data and sophisticated algorithms, which 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, and 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 not only real-time but also live.

In this paper, we first 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 a unique object. This method provides good quality for new views by using only one computer. Most of our tests were computed from four 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 ten or more. Only the four 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.

Finally, we present an adaptation of our method that can create on-line multiple new views simultaneously. Our method is especially designed for autostereoscopic displays and can provide up to 16 images in real-time from 4 input webcams connected to a consumer grade computer.

In the following parts, we first propose a survey of the latest off-line and on-line VBR methods. Then, we explain the plane-sweep algorithm, our contribution related to the rendering, and our camera array set up. We also detail our implementation and present our experimental results, followed by how our method can be modified to create multiple views of the scene in real-time. Finally, we present additional experimental results with this multiple view method.

## 2 Previous work

This section surveys previous work on both recent off-line and on-line VBR techniques.

### 2.1 Off-line video-based rendering

The first proposed VBR method is the virtualized reality presented by Kanade et al.[1]. 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.[2] and Zitnick et al.[3] followed the same approach. Goldlucke et al.[2] used 100 cameras and created new views of the scene in real-time. Zitnick et al.[3] provided high quality images in real-time using 8 cameras. The depth maps are computed using a segmentation method and the

Manuscript received December 25, 2007; revised March 18, 2008.
This work was supported by Foundation of Technology Supporting the Creation of Digital Media Contents project (CREST, JST), Japan.
*Corresponding author: nozick@ozawa.ics.keio.ac.jp

rendering is performed with a layered image representation. The stanford camera array presented by Wilburn et al.[4] computes an optical flow instead of a depth-map and provides real-time rendering from 100 cameras. Franco and Boyer[5] provided new views from six 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 cannot expect the on-line methods to have 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 3D 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 [6]. The most accurate on-line visual hulls method seems to be the image-based visual hulls presented by Matusik et al.[7] 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 methods proposed by Li et al.[8,9] are 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 make their use more difficult.

Another possibility to achieve on-line rendering is to use a distributed light field as proposed by Yang et al.[10] They presented 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 requested new view. 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 a graphic hardware, graphics processing unit (GPU). The plane-sweep algorithm introduced by Collins[11] was adapted to on-line rendering by Yang et al.[12] They computed new views in real-time from five cameras using four computers. Geys et al.[13] also used a plane-sweep approach to find out the scene geometry and rendered new views in real-time from three cameras and one computer. Since our method belongs to the latter family, we will expose the basic plane-sweep algorithm and contribution in the next section. Then, we will detail our method.

## 3 Plane-sweep algorithm

This section exposes the basic plane-sweep algorithm and surveys the 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 Fig. 1.

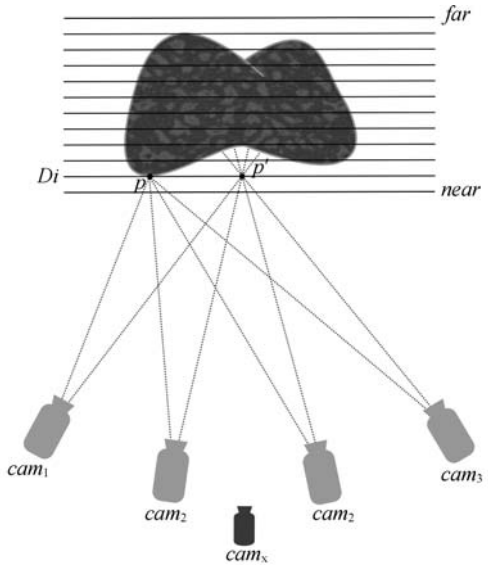


Fig. 1 Plane-sweep: Guiding principle

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 the visible object. This point will probably not be seen by the input cameras with the same color. Fig. 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 in Fig. 2. Then, the computed scores and colors are projected onto the virtual camera $cam_x$. The virtual view is hence updated in a $z$-buffer style: the color and score (assimilated to depth in a $z$-buffer) of the 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 in Fig. 3. Then, the next plane $D_i$ is computed. The final image is obtained when every plane is computed.

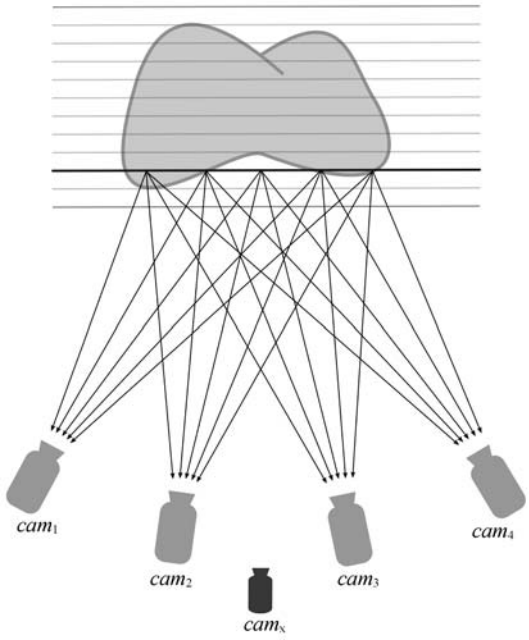


Fig. 2 Every point of the current plane is projected onto the input images (A score and a color are computed for these points according to the matching of the colors found.)

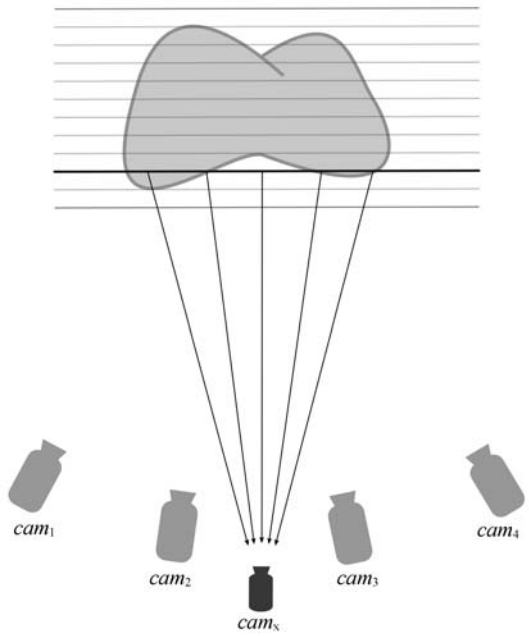


Fig. 3 The computed scores and colors are projected onto the virtual camera

### 3.2 Previous implementations

Yang et al.[12] proposed an implementation of the plane-sweep algorithm using register combiners. The system chooses 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 sum of squared difference (SSD). 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 $cam_x$. 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[13] 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 used scoring method is similar to the method proposed by Yang et al.[12], but it only computes a depth map. Then, an energy minimization method based on a graph cut algorithm on the 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 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 compute scores by comparing input images with the reference image. We propose a method that avoids the use of such a reference image that may not be representative of the virtual view. Our method also uses every input image together rather than computing 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 $c_j$. 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 $c_j$. Indeed, the average of similar colors is very representative of the colors set. However, the average color computed from mismatching colors will not be a valid color for our method since these colors provide a low score, and this average color will likely not be selected for the virtual image computation. This plane-sweep implementation is summarized in Algorithm 1.

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.

**Algorithm 1.**

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 $p$: $color_p = \frac{1}{n}\sum_{j=1\cdots n} c_j$
- compute the score of $p$: $score_p = \sum_{j=1\cdots n}(c_j - color)^2$

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.

## 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 the used input images since the score computation time linearly depends on the number of input images. Furthermore, real-time video stream control requires special devices when the number of cameras is large.

We propose a webcam management to handle up to ten or more USB cameras from a single computer (see Fig. 4). Considering the position of the virtual camera, the system selects the four most appropriate cameras that are used to compute the new view, and the video streams from non-selected cameras are disabled. Then, for the next view, if the virtual camera moves, the set of the 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.

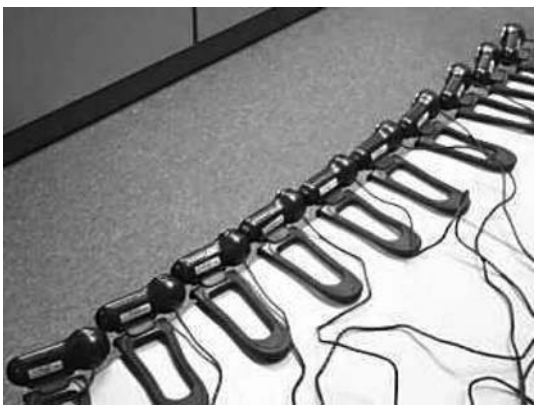

Fig. 4 Ten webcams connected to a laptop via a USB hub

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 used 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. Figs. 5 and 6 depict these two possibilities in an aligned configuration. In a circle arc configuration, using eight webcams rather than four 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 significantly increase.

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


Fig. 5 The additional cameras are used to extend the range of available virtual view points


Fig. 6 The additional cameras are used to increase the visual quality of the new view by using a dense cameras disposition

## 6 Implementation

Since our webcams have a fixed focal length, we compute accurately their internal parameters using Zhang′s calibration[14]. Then, we can freely move them for our experimentation 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 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[6,pp.23] or by the method presented by Yamamoto and Oi[15]. 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 by Geys et al.[13]

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 in [16]. 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 due 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 be 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 an nVidia GeForce 7400 TC. The video acquisition was performed with USB Logitech fusion webcams connected to the computer via a USB hub. With a 320×240 resolution, 4 webcams streaming simultaneously provided 15 frames per second. Our tests showed that for this resolution and frame rate, avoiding camera synchronization does not affect the visual result.

The computation time to create a new view was linearly dependent on the number of planes used, the number of input images, and 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 experimentation, we used 60 planes to ensure an optimal visual quality. We set the virtual image resolution (output image) to be 320×240. With such a configuration, the number of input cameras was limited to 4 due to the GPU computation time. Our method reached 15 frames per second. Fig. 7 shows the real-view taken exactly between two adjacent cameras, the corresponding created image, and the difference. This difference is small enough to ensure good quality result.

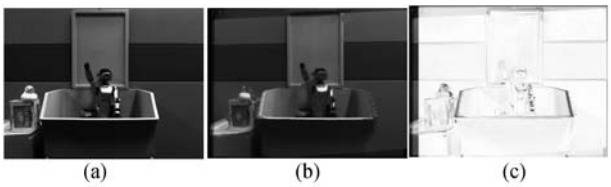


Fig. 7 (a) Real view; (b) Computed view with the virtual camera between the two adjacent input cameras; (c) Difference between the real and computed views

As shown in Fig. 8, using ten webcams provides a large

Fig. 8 Each new view is computed on-line with a laptop using 4 cameras selected among 10 (The scene was discretized with 60 planes and this method reaches 15 fps.)

range of available virtual view points. Hence, because of our webcam management method, the range of available virtual view points can be extended without any visual quality loss. The user can decrease the base-line between the cameras. Fig. 9 shows the two images created from 4 webcams. In the first case, the distance between the two adjacent cameras is 10 cm, in the second case, 5 cm. We can see that reducing the base-line greatly improves the visual result.


Fig. 9 Images created from 4 webcams. (a) 10 cm base-line; (b) 5 cm base-line view

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

In our tests, the bottle neck was 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.

# 8 Multiple view computation and video-based rendering

The previous sections detailed our plane-sweep algorithm and presented some real-time results. 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 viewpoints 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.

In the following parts, we introduce autostereoscopic devices mechanism and their consequences on the video stream acquisition. We also present the main issues for VBR methods to satisfy these constraints. Then, we detail how our plane-sweep algorithm can be modified to create multiple views in real-time.

## 8.1 Autostereoscopic display

In recent years, stereoscopic technology has advanced from stereoscopic to autostereoscopic displays. The latter family does not involve any glasses or specific device for the user. Such a screen displays several images of the same scene and provides to the user an adequate stereoscopic view according to his relative position from the screen. Moreover, these displays can support multi-user applications. Currently, commercial autostereoscopic displays are available from several famous companies. The spatial multiplex system is a common method to provide stereoscopic images. A lenticular sheet is laid on the screen such that every lens covers a group of pixels. According to the users′ position, the lens will show the adequate pixels. The major part of commercial autostereoscopic displays requires around 10 views. Some recent devices can display up to 64 views[17]. More information about autostereoscopic displays can be found in [18].

Computer graphics applications[19] for autostereoscopic devices already exist but stereoscopic video of real scenes remains a problem if the display supports more than six or seven views. Indeed, using one camera per view involves a huge quantity of data and hence storage and transfer issues. VBR methods are an alternative to decrease the number of input cameras required for autostereoscopic devices, however, few VBR methods are suited to on-line multiple view rendering.

## 8.2 Video-based rendering and stereovision

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 proposed by Li et al.[8,9] may run on a single computer but the ability to compute several images simultaneously should be demonstrated. Furthermore, the visual hulls method is suited for an all around camera configuration but not for a dense aligned camera configuration required for autostereoscopic display applications. Finally, the visual hulls involves a background extraction, thus only the main objects can be rendered.

The distributed light field proposed by Yang et al.[10] 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 due to the space decomposition using planes. Indeed, scores and colors computed on every plane represent a 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.

## 8.3 Practical application

Some companies already proposed a 3D autostereoscopic display for mobile phones designed to display a 3D menu, images and computer graphics videos[20]. This paper proposes a practical application of our method on communi-

cations between a mobile phone and a computer for real scenes. Such situation occurs when someone calls his family or company. The computer side provides multiple images of its environment to the mobile phone user.

The main restriction of our application concerns the bandwidth limitation during the video stream transfer from the computer to the mobile phone. However, mobile phone technology can easily support standard online video decompression. Moreover, the bandwidth issue is lessened by the low resolution of the mobile phone screen. Finally, the system providing the views should work on a consumer grade computer that is attractive for commercial applications. Using ten or more webcams can provide enough views for a stereoscopic display, but even with a low resolution, real-time video-stream acquisition is a serious issue with a single computer.

# 9 Multiple view rendering

## 9.1 Multiple view computation

Due to its plane decomposition of space, our single view plane-sweep method can be modified to a $k+1$ passes algorithm, where $k$ is the number of virtual cameras, to provide on-line multiple new views.

For every plane $D_i$ (see Fig. 1), the score and color of every point is computed in the first pass. This pass is absolutely independent of the number of virtual views to create. The information computed during this pass is then projected onto every virtual view in $k$ passes. During the 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 described in Algorithm 2.

**Algorithm 2.**

Reset the scores and colors of the virtual cameras′ memory $V_j$ $(j \in \{1, \cdots, k\})$

**for** each plane $D_i$ from *far* to *near*

**for** each point (fragment) $p$ of $D_i$

- compute a score *color* and a color *score*
- store *color* and *score* in an array $T(p) = (color, score)$

**for** each virtual 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.

Like in the single view method, the score and color are computed only once for every point of each plane. Since the projection of the information on every virtual view differs, the final views will be different. These information projections are very fast compared to the score and color computation. Hence, sharing the score and color computation speeds up the application and avoids the redundancy process without any loss of visual quality.

## 9.2 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 views that can be computed. We propose a method where every process is done off-screen using a frame buffer object. Red, green, blue, alpha (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 cameras texture will replace the frame buffer used on the single view method. As illustrated on Fig. 10 (a), the score and color computation of a plane does not differ from the single view method except that the rendering is performed on texture. Naturally, the rendering has to be associated with a projection matrix. We select the central virtual camera as the reference camera for this projection (Fig. 10 (b)).

Then, every virtual camera involves an additional rendering pass. During the pass, the score and color texture is projected onto the current plane using the reference camera projection matrix (Fig. 10 (c)). The textured plane is then projected onto the virtual camera (Fig. 10 (d)) using fragment shaders. The texture associated with 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. After computating for the last plane, the virtual camera texture can be extracted to provide new images of the virtual views.

## 9.3 Compression and image transfer

Since 3D display and 3D video broadcasting services became feasible, 3D video data compression has become an active research field. Indeed, without any compression, the transmission bandwidth linearly increases with the number of views and becomes a severe limitation for the display frame-rate. Nevertheless, stereoscopic views represent the same scene and contain a huge amount of redundancies. Thus, the basic concept of 3D video compression is to remove these redundancies among the views. There exist several stereoscopic compression methods. For more information, the reader can refer to Kalva et al.[21].

Since we want our system to be used with mobile phones, the problem is a bit different. The screen resolution is lower than that for standard 3D displays but the available bandwidth is also restricted by the mobile phone communication system. Furthermore, the compression part achieved by the computer should be fast and should not require too many CPU capabilities. In our tests, we chose an MPEG2 compression. Indeed, the views to be transferred consisted of the input images and the virtual images. These views can be sorted by position (from left to right for example) such that they become suited to be compressed with a standard video compression method. Such compression is performed

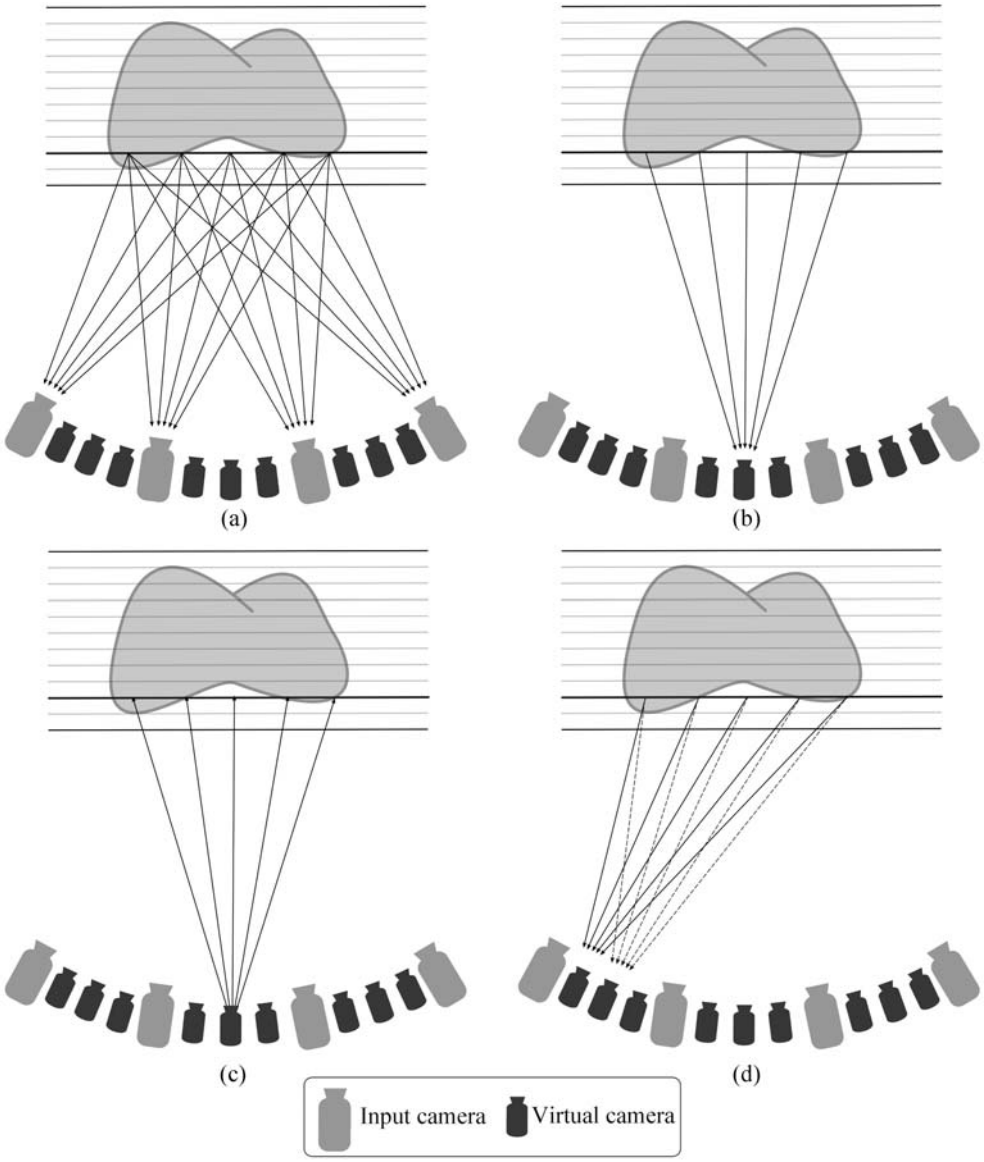


Fig. 10 (a) The points of the current plane are projected onto every input camera to read the corresponding color; (b) The colors found are used to compute a score and a color during a rendering process on the reference camera; (c) For every virtual camera, the scores and colors are projected onto the current plane using the reference camera projection matrix; (d) The scores and colors are projected from the current plane to the virtual cameras

by the CPU, and hence is compatible for real-time computation with our VBR method which mainly uses the GPU. In addition, MPEG2 decompression is not a problem with mobile phone hardware.

## 9.4 Results

We have implemented our system on an Intel Core Duo 1.86 GHz PC with an nVidia GeForce 7900 GTX. The video acquisition is performed by 4 USB Logitech fusion webcams connected to the computer via a USB hub. With a 320× 240 resolution, the acquisition frame rate reaches 15 frames per second.

In our tests, we set the output image resolution to 320× 240. Since our system is designed for stereoscopic display, the base-line between extreme cameras is restricted. Like for the tests performed for the single view method, we use 60 planes to ensure an optimal visual quality. Our tests include image compression and transfer. Since we considered that the data transfer should be done by the mobile phone operator, we just tested our compressed video transfer with a UDP network protocol with another PC. There exist more powerful tools for such video streaming but this is not the main purpose of our paper. The video compression is done by an MPEG2 method and reaches a 1:41 compression rate. Thus, the transferred data is highly compressed and well suited to be decompressed by mobile phones.

The number of virtual views depends on the application. In our case, we tested our system with 6, 9, 12, 15, and 18

Fig. 11 Sample result of 16 views: 12 virtual views and 4 input images (on the diagonal) (These images have been computed using 60 planes at 8.7 frames per second. Parallel-eyed viewing provides stereoscopic images.)

virtual cameras set between adjacent input cameras. The speed results obtained with such configuration are shown in Table 1. This computation includes compression and transfer of both virtual views and input images. Table 1 also includes the frame rate of the classic method which computes independently every virtual view.

Table 1 Frame rate and number of virtual views

| Number of virtual views | Number of total views | Frame rate (fps) | Classic method (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 |

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 times faster for 18 virtual views without any loss of quality.

Fig. 11 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 that decreases the imperfection impact. As shown in Fig. 11, 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.

## 10 Conclusion

This paper presents an on-line VBR 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, while other methods compute images by pair. In addition, we present a video stream management that extends the number of potential webcams used to create a new view. This technique involves a better flexibility of the cameras′ position and increases the visual result. Compared to other on-line VBR techniques, this method can handle the scene background, does not require more than one computer and

provides high quality images.

We also present an adaptation of our plane-sweep 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 on mobile phones communicating with a computer. The use of only one computer and few webcams makes this system low cost and well suited for commercial applications, particularly for the latest mobile phone autostereoscopic displays that require more than 15 images per frame. Based on our knowledge, no other VBR method exists that provides equivalent result with such a configuration.

Concerning other extensions of this method, we believe that our multiple-view system can be easily adapted for multi-user stereoscopic teleconference applications. The system would work as a server that provides stereoscopic views for several clients from desired viewpoints. We also intend to achieve an optimization on the plane repartition in order to increase the visual quality without adding any further planes.

## References

[1] T. Kanade, P. J. Narayanan, P. Rander. Virtualized Reality: Concepts and Early Results. In *Proceedings of IEEE Workshop on Representation of Visual Scenes*, IEEE Press, Cambridge, USA, pp. 69–76, 1995.

[2] B. Goldlucke, M. A. Magnor, B. Wilburn. Hardware Accelerated Dynamic Light Field Rendering. In *Proceedings of Vision, Modeling and Visualization*, Erlangen, Germany, pp. 455–462, 2002.

[3] C. L. Zitnick, S. B. Kang, M. Uyttendaele, S. Winder, R. Szeliski. High-quality Video View Interpolation. *ACM Transactions on Graphics*, vol. 23, no. 3, pp. 600–608, 2004.

[4] B. Wilburn, N. Joshi, V. Vaish, E. V. Talvala, E. Antunez, A. Barth, A. Adams, M. Horowitz, M. Levoy. High Performance Imaging Using Large Camera Arrays. *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 765–776, 2005.

[5] J. S. Franco, E. Boyer. Fusion of Multi-view Silhouette Cues Using a Space Occupancy Grid. In *Proceedings of the 10th IEEE International Conference on Computer Vision*, IEEE Press, Beijing, China, vol. 2, pp. 1747–1753, 2005.

[6] M. A. Magnor. *Video-based Rendering*, A K Peters Ltd, 2005.

[7] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, L. McMillan. Image-based Visual Hulls. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press, NY, USA, pp. 369–374, 2000.

[8] M. Li, Magnor, H. P. Seidel. Hardware-accelerated Visual Hull Reconstruction and Rendering. In *Proceedings of Graphics Interface*, A K Peters, Ltd., Halifax, Canada, pp. 65–71, 2003.

[9] M. Li, M. A. Magnor, H. P. Seidel. Online Accelerated Rendering of Visual Hulls in Real Scenes. *Journal of WSCG*, vol. 11, no. 2, pp. 290–297, 2003.

[10] J. C. Yang, M. Everett, C. Buehler, L. McMillan. A Real-time Distributed Light Field Camera. In *Proceedings of the 13th Eurographics Workshop on Rendering*, Eurographics Association Aire-la-Ville, Pisa, Italy, pp. 77–86, 2002.

[11] R. T. Collins. A Space-sweep Approach to True Multi-image Matching. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Press, San Francisco, USA, pp. 358–363, 1996.

[12] R. Yang, G. Welch, G. Bishop. Real-time Consensus-based Scene Reconstruction Using Commodity Graphics Hardware. In *Proceedings of the 10th Pacific Conference on Computer Graphics and Applications*, IEEE Press, Beijing, China, pp. 225–234, 2002.

[13] I. Geys, S. D. Roeck, L. V. Gool. The Augmented Auditorium: Fast Interpolated and Augmented View Generation. In *Proceedings of the 2nd IEE European Conference on Visual Media Production*, IEEE Press, London, UK, pp. 94–103, 2005.

[14] Z. Y. Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[15] K. Yamamoto, R. Oi. Color Correction for Multi-view Video by Minimizing Energy of View Networks. In *Proceedings of ACCV07 Wokrshop on Multi-dimensional and Multi-view Image Processing*, Tokyo, Japan, pp. 9–16, 2007.

[16] M. Pharr, R. Fernando. *GPU Gems 2: Programming Techniques for High-performance Graphics and General-purpose Computation*, Addison-Wesley Professional, Har/Cdr edition, 2005.

[17] Y. Takaki. High-density Directional Display for Generating Natural Three-dimensional Images. *Proceedings of the IEEE*, vol. 94, no. 3, pp. 654–663, 2006.

[18] N. A. Dodgson. Autostereoscopic 3D Displays. *Computer*, vol. 38, no. 8, pp. 31–36, 2005.

[19] Y. Morvan, D. Farin, P. H. N. De With. Multiview Depth-image Compression Using an Extended H.264 Encoder. In *Proceedings of Advanced Concepts for Intelligent Vision System, Lecture Notes in Computer Science*, Springer, vol. 4678, pp. 675–686, 2007.

[20] J. Harrold, G. J. Woodgate. Autostereoscopic Display Technology for Mobile 3DTV Applications. In *Proceedings of the SPIE, Stereoscopic Displays and Applications XVIII*, Andrew J. Woods, Neil A. Dodgson, John O. Merritt, Mark T. Bolas, Ian E. McDowall (eds.), vol. 6490, 2007.

[21] H. Kalva, L. Christodoulou, L. Mayron, O. Marques, B. Furht. Challenges and Opportunities in Video Coding for 3D TV. In *Proceedings of IEEE International Conference on Multimedia and Expo*, IEEE Press, Canada, pp. 1689–1692, 2006.

**Vincent Nozick** received his Maitrise, DEA, and Ph. D. degrees in computer sciences from Université de Marne-la-Vallée, Champs sur Marne, France, in 2001, 2002, and 2006, respectively. He was a visiting researcher in Hideo Saito Laboratory, Keio University, Yokohama, Japan, from 2006 to 2007. He has served as an assistant professor at Hideo Saito Laboratory since 2007. He was an MNRT fellow from the French Research and Education Ministry from 2002 to 2005 and ATER from 2005 to 2006. He was a Lavoisier fellow from the French Foreign Ministry from 2006 to 2007.

His research interests include computer graphics, computer vision, and video-based rendering.

**Hideo Saito** received his B. Eng. , M. Eng. , and Ph. D. degrees in electrical engineering from Keio University Yokohama, Japan, in 1987, 1989, and 1992, respectively. He has been on the faculty of Department of Electrical Engineering, Keio University, since 1992. From 1997 to 1999, he was a visiting researcher at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, where he joined the virtualized reality project. From 2000 to 2003, he was also a researcher with PRESTO, JST. Since 2006, he has been a professor at the Department of Information and Computer Science, Keio University. He is a member of IEICE, IPSJ, VRST, ITE, IIEEJ, and SICE.

His research interests include computer vision, image processing, augmented reality, and human-computer interaction.