Online Multiple View Computation for Autostereoscopic Display

Vincent Nozick and Hideo Saito

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

Abstract. This paper presents a new online Video-Based Rendering method that creates simultaneously multiple views at every new frame. Our system is especially designed for communication between mobile phones using autostereoscopic display and computers. The new views are computed from 4 webcams connected to a computer and are compressed in order to be transfered to the mobile phone. Thanks to GPU programming, our method provides up to 16 images of the scene in real-time. The use of both GPU and CPU makes our method work on only one consumer grade computer.

Keywords: video-based rendering, autostereoscopic, view interpolation.

1 Introduction

In recent years, steoroscopic technology has advanced from stereoscopic to autostereoscopic displays. This 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. This ability makes autostereoscopic displays very convenient to use, especially for multi-user applications. Autostereoscopic displays can have various applications like 3D TV, games, 3D teleconference or medical applications. In this paper, we will focus on the communication between mobile phones using 3D display and computers using cameras. Indeed, mobile TV is now a commercial reality and the next mobile phone evolution will include 3D display.

Autostereoscopic displays require multiple views of the scene at every frame. Stereoscopic animations or menus can easily be achieved by computer graphics methods. 3D content can also be generated from videos of real-scenes using several cameras. However this approach is not suited for mobil e phone applications due to their restricted bandwidth and low capacity to use complex decompression algorithms. Furthermore, systems using more than 10 cameras will probably not be attractive for commercial applications. Video-Based Rendering (VBR) methods can provide new views of a scene from a restricted set of videos and thus decrease the number of required cameras. Nevertheless few VBR methods provide online rendering and most of these methods are not suited for

D. Mery and L. Rueda (Eds.): PSIVT 2007, LNCS 4872, pp. 399–412, 2007.

multiple image rendering. Indeed, autostereoscopic applications require several new images simultaneously and most of VBR method should compute these new views independently, decreasing the frame rate.

This article present a new VBR algorithm that can create online multiple new views simultaneously using GPU programing. This method requires 4 webcams connected to a consumer grade computer and can provide up to 16 images in real-time, including compression and network transfer. In the following parts, we introduce recent autostereoscopic devices and propose a survey of the latest online VBR methods. Then we explain the plane sweep algorithm and our contribution. Finally, we detail our implementation and we present experimental results.

2 Autostereoscopic Displays

A stereoscopic display requires at least two views of the same scene to create depth perception. During more than a century, the three most popular methods have been anaglyph, polarization and shutter methods [1]. These three techniques involve the use of adapted glasses. Recent research on 3d display technologies made stereoscopic system advance from stereoscopic to autostereoscopic displays. This latter category presents tree major advantages. First, users do not need any special devices like glasses. Second, these displays can provide more than two views simultaneously. The user receives an adequate pair of stereoscopic images according to his position from the screen. Finally, these displays can support multi-user applications.

Currently, commercial autostereoscopic displays are available from several famous companies. Spatial multiplex system is a common method to provide stereoscopic images. A lenticular sheet is laid on the screen such every lens covers a group of pixels. According to the user's position, the lens will show the adequate pixel. The major part of commercial autostereoscopic displays requires around 10 views. Some recent devices can display up to 60 views. More informations about autostereoscopic displays can be found on [2].

The main purpose of stereoscopic display is to increase the user's immersion sensation. Stereoscopic display applications includes scientific visualization, medical imaging, telepresence or gaming. In this paper, we will focus on mobile phone applications. Indeed, some companies already propose 3D autostereoscopic display for mobile phone designed for display 3D menu, images and videos. This paper especially focuses on communications between a mobile phone and a computer for real scenes. Such situation occurs when someone call his family or company. The computer side provides multiple images of its environment to the mobile phone user. Harrold and Woodgate [3] describe such device and present a method to transfer and render computer graphics stereoscopic animations. However stereoscopic video of real scenes remains a problem if the display supports more than 6 or 7 views. Indeed, using one camera per view involves a huge quantity of data and hence storage and transfer issues.

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 to be attractive for commercial applications. Using 10 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.

In the following parts, we propose an online video-based rendering method that provides multiple images of the same scene from a small set of webcams using only one consumer grad computer.

3 Online Video-Based Rendering

Given a set a videos taken from video cameras, Video-Based Rendering methods provides new views of the scene from new viewpoint. Hence these methods are well suited to reduce the number of input cameras for autostereoscopic display systems. VBR methods are divided into two families : off-line and online methods. Off-line methods focus on the visual quality rather than on the computation time. They first calibrate the cameras and record the video streams. Then they compute these videos to extract scene informations. The rendering step can start only when the data computation is completed. Most of the off-line methods provide real-time rendering but are not suited for live rendering. On the other hand, online method are fast enough to record, compute and render a new view in real-time however few VBR methods reach online rendering. Finally, autostereoscopic display applications require not only online VBR methods but also methods that can create simultaneously several new views of the scene for every frame.

The most popular online VBR method is the Visual Hulls algorithm. This method extracts the silhouette of the main object of the scene on every input image. The shape of this object is then approximated by the intersection of the projected silhouettes. There exist several online implementations of the Visual Hulls described in [5]. The most accurate online Visual Hulls method seems to be the Image-Based Visual Hulls presented by Matusik et al. [6]. This method creates news views in real-time from 4 cameras. Each camera is controlled by one computer and an additional computer create the new views. The methods proposed by Li et al. [7,8] 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. Finally, the Visual Hulls algorithm requires a background extraction, thus only the main “objects” can be rendered.

Another possibility to reach online rendering is to use a distributed Light Field as proposed by Yang et al. [9]. 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. Thus this method is incompatible with a commercial use of stereoscopic applications.

Finally, some Plane-Sweep methods reach online rendering using graphic hardware (GPU). The Plane-Sweep algorithm introduced by Collins [10] was adapted to online rendering by Yang et al. [11]. They compute new views in real-time from 5 cameras using 4 computers. Geys et al. [12] also use a Plane-Sweep approach to find out the scene geometry and render new views in real-time from 3 cameras and one computer. The Plane-Sweep algorithm is effective when the input cameras are close to each other and hence is highly capable with an aligned camera configuration. Since our method follows a Plane-Sweep approach, we will expose the basic Plane-Sweep algorithm in the next section. Then we will detail our method for both single and multiple new views creation.

4 Single View Computation

In this section, we present the Plane-Sweep algorithm and [11,12] contribution. Then we detail our new scoring method.

4.1 Plane-Sweep Algorithm Overview

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

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

During the new view creation process, every plane D_i is computed in a back to front order. Each pixel p of a plane D_i is projected onto the input images. Then, a score and a representative color are computed according to the matching of the colors found. A good score corresponds to similar colors. This process is illustrated on Figure 2. Then, the computed scores and colors are projected on the virtual camera cam_x. The virtual view is hence updated in a z-buffer style : the color and score (depth in a z-buffer) of a pixel of this virtual image is updated only if the projected point p provides a better score than the current score. This

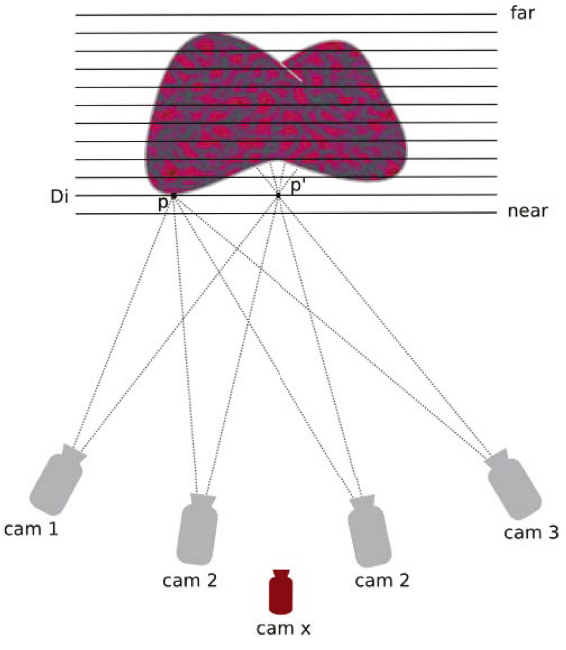

Fig. 1. Plane-Sweep : guiding principle

process is depicted on Figure 2. Then the next plane D_{i+1} is computed. The final image is obtained when every plane is computed.

4.2 Scoring Stage

Yang et al. [11] propose 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 SSD (Sum of Squared Difference). The final score of p is the sum of these SSD.

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

Geys et al.'s method [12] begins with a background extraction. The background geometry is supposed to be static. This assumption restricts the application of the Plane-Sweep algorithm to the foreground part. The scoring method used is similar to the method proposed by Yang et al. but they only compute

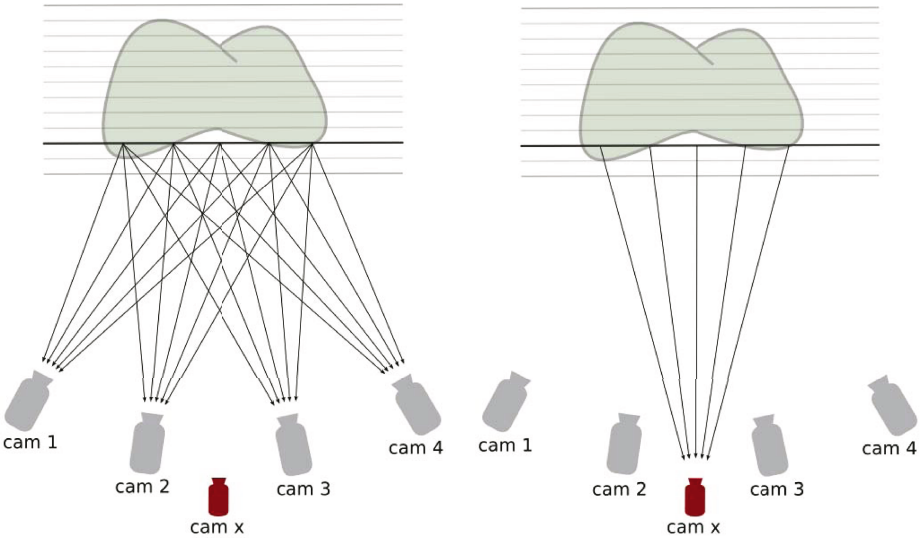

Fig. 2. *Left* : Every point of the current plane is projected on the input images. A score and a color are computed for these points according to the matching of the colors found. *Right* : The computed scores and colors are projected on the virtual camera.

a depth map. Then, an energy minimization method based on a graph cut algorithm 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 online rendering using 3 cameras and only one computer. However, the background geometry must be static.

Our main contribution to the Plane-Sweep algorithm concerns the score computation. Indeed, this operation is a crucial step since both visual results and time 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 image that may not be representative of the virtual view. Our method also use every input image together rather than to compute images by pair.

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

This plane sweep implementation can be summarized as follows :

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

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

5 Multiple View Computation

A basic approach to render multiple views would be to compute every virtual view independently. However most of online 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 decompostion 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.

Therefore, our single view Plane-Sweep method can be modified in a $k + 1$ passes algorithm, where k is the number of virtual cameras. For every plane D_i, 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' memory V_j $(j \in \{1, ..., k\})$
- **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 plan. Since the projection of these informations 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 redundancy process without any loss of visual quality.

6 Implementation

Since our webcams have a fixed focal length, we can compute accurately their internal parameters using Zhang calibration [14]. Then we can freely move them for our experimentations and only a single view of a calibration chessboard is required to perform a full calibration. Color calibration can be performed by the method proposed by Magnor [5, page 23]. This method is effective only for small corrections.

We usually set the far plane as the calibration marker 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. [12]. We use OpenGL for the rendering part. For each new view, we perform a first 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 [16].

During the score and color computation, each plane D_i is drawn as a 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 described in section 4.

For the single view method, 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.

The use of the 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. As illustrated on Figure 3 (a), 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 (Figure 3 (b)). Then, every virtual camera involves an additional rendering pass. During a pass, the score and color texture is projected on the curent plane using the reference camera projection matrix (Figure 3 (c)). The textured plane is then projected on the virtual camera (Figure 3 (d)) using fragment shaders. The texture associated to the curent virtual camera is used for both rendering and reading the last selected scores and colors. The fragment program decides to update a fragment information or to keep the current texture value according to the last selected scores as described in section 5. After the last plane computation, the virtual camera's texture can be extracted as images of the virtual views.

The computation time linearly depends on the number of planes used, on the number of virtual cameras and on the output images resolution. The number of input cameras has both a repercussion on the image transfer from the main memory to the GPU and on the score computation performances. Most of the computation is done by the graphic card, hence the CPU is free for the video stream acquisition, virtual views compression and transfer.

7 Compression and Images Transfert

Since 3d display and 3d video broadcasting services became feasible, 3d video data compression has been 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 informations, the reader can refer to Kalva et al. [17].

Since we want our system to be used with mobile phones, the problem is a bit different. The screen resolution is lower than 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 a MPEG2 compression. Indeed, the views to be transfered consist of the input images and the virtual images. These views can be sorted by position (from left to right for example) such they become suited to be compressed with a standard video compression method. Such compression is performed by the CPU and

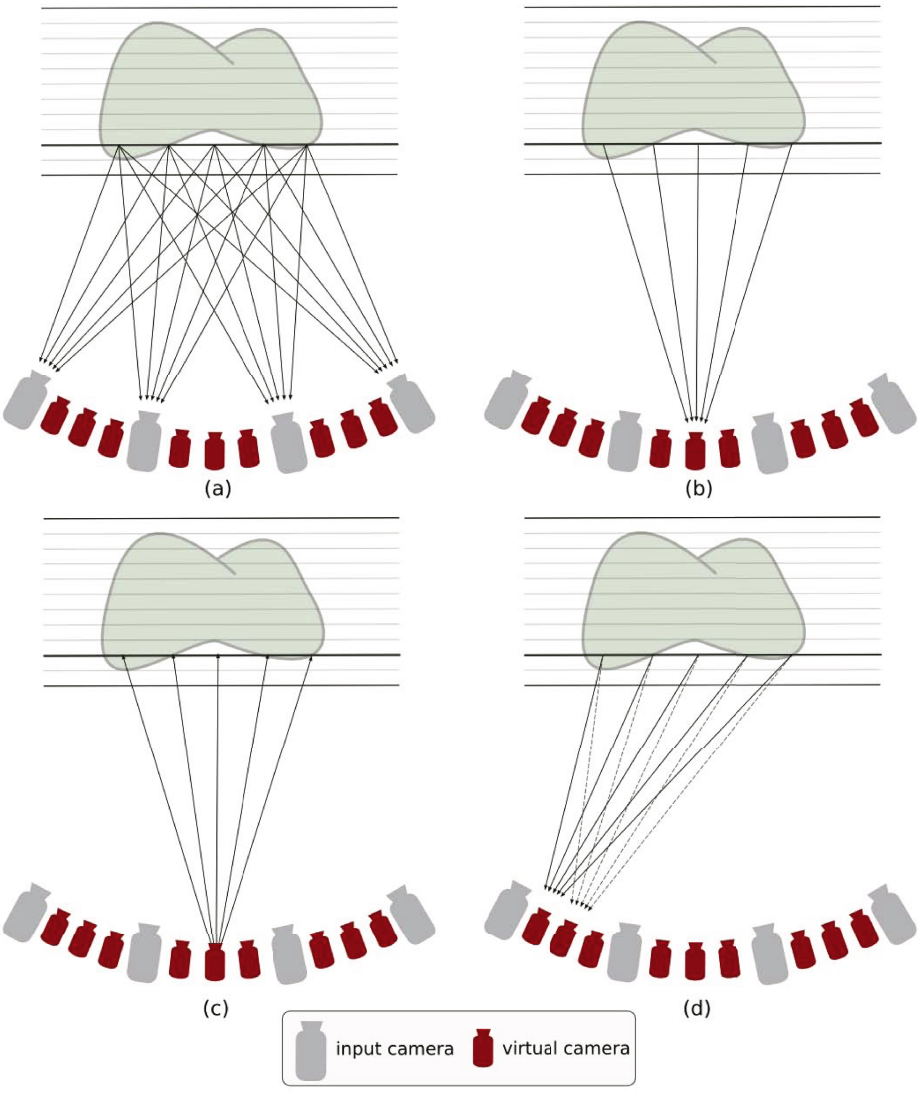

Fig. 3. (a) : The points of the current plane are projected on every input camera to read the corresponding color. (b) : The colors found are used to compute a score and a color during a rendering process on the reference camera . (c) and (d) : For every virtual camera, the scores and colors are projected on the current plane using the reference camera projection matrix (c). The scores and colors are projected from the current plane to the virtual camera (d).

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.

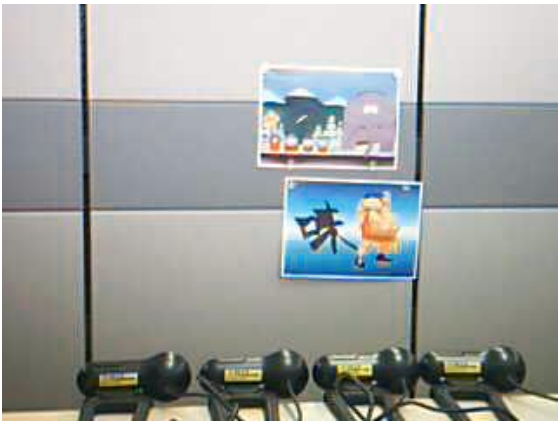

Fig. 4. Cameras configuration

The compressed images are then transfered to the user. Since we consider that the data transfer should be done by the mobile phone operator, we just tested our compressed video transfer with an UDP network protocol with another PC. There exist more powerful tools for such video streaming but this is not the main purpose of our article.

8 Results

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

As explained in part 6, the computation times depends among others, on the number of planes, on the number of virtual cameras and on the virtual view resolution. 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 camera is restricted. In such condition, our tests shown that under 10 planes, the visual results becomes unsatisfactory and using more that 60 planes does not improve the visual result. Hence, we used 60 planes in our experimentation to ensure an optimal visual quality.

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 obtains with such configuration are shown on 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 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.

Figure 5 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 2 views per user (right and left eyes) and the fusion of the two images decreases the imperfection impact. As shown on Figure 5, 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.

In our tests, we compressed and send the images to a client computer. The compression is done by a MPEG2 method and reaches a 1:41 compression rate. Thus the transfered data is highly compressed and well suited to be decompressed by mobile phones.

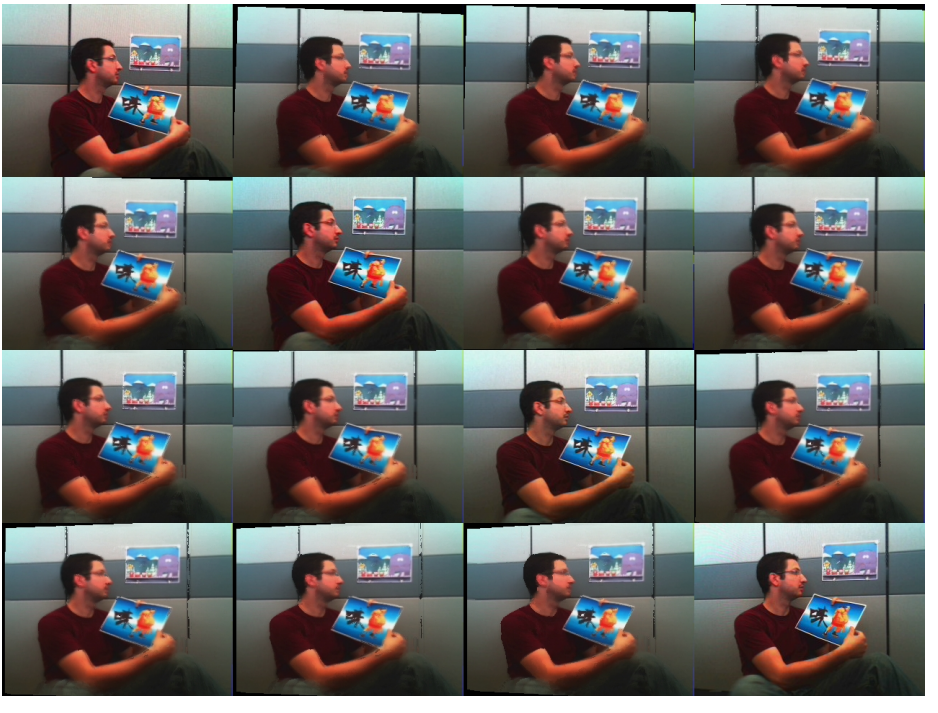

Fig. 5. Sample result of 16 views : 12 virtual views and 4 input images on the diagonal. These images have been computed using 60 planes at 8.7 frames per second. Parallel-eyed viewing provides stereoscopic images.

Table 1. frame rate and number of virtual views

number of virtual views	number of total views	frame rate (frames per second)	classic method (frames per second)
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

9 Conclusion

This article presents a live video-based rendering method that provides simultaneous multiple views of a scene from a small set of webcams. We propose a new scoring method that provides good visual quality images in real-time thanks to fragment shaders. 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 online 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 that 15 images per frame. According to our knowledge, there does not exist other VBR method that provides equivalent result with such configuration.

Concerning other extensions of this method, we believe that our multiple-view system can be easily adapted for multi-users stereoscopic teleconference applications. The system would work as a server that provides stereoscopic views for several clients from desired viewpoints.

Acknowledgment

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

References

1. Okoshi, T.: Three-Dimensional Imaging Techniques. Academic Press, San Diego (1977)
2. Dodgson, N.A.: Autostereoscopic 3D Displays. Computer 38(8), 31–36 (2005)
3. Harrold, J., Woodgate, G.: Autostereoscopic display technology for mobile 3DTV applications. In: Proc. of the SPIE, vol. 6490 (2007)
4. Goldlucke, B., Magnor, M.A., Wilburn, B.: Hardware accelerated Dynamic Light Field Rendering. Modelling and Visualization VMV 2002, Germany, 455–462 (2002)

5. Magnor, M.A.: Video-Based Rendering. A K Peters Ltd (2005)
6. Matusik, W., Buehler, C., Raskar, R., Gortler, S.J., McMillan, L.: Image-Based Visual Hulls. ACM SIGGRAPH 2000, 369–374 (2000)
7. Li, M., Magnor, M.A., Seidel, H.P.: Online Accelerated Rendering of Visual Hulls in Real Scenes. In: International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG 2003), pp. 290–297 (2003)
8. Li, M., Magnor, M.A., Seidel, H.P.: Hardware-Accelerated Visual Hull Reconstruction and Rendering. Graphics Interface GI 2003, Canada, 65–71 (2003)
9. Yang, J.C., Everett, M., Buehler, C., McMillan, L.: A real-time distributed light field camera. In: 13th Eurographics workshop on Rendering, Italy, pp. 77–86 (2002)
10. Collins, R.T.: A Space-Sweep Approach to True Multi-Image. Computer Vision and Pattern Recognition Conf., 358–363 (1996)
11. Yang, R., Welch, G., Bishop, G.: Real-Time Consensus-Based Scene Reconstruction using Commodity Graphics Hardware. Pacific Graphics, 225–234 (2002)
12. Geys, I., De Roeck, S., Van Gool, L.: The Augmented Auditorium: Fast Interpolated and Augmented View Generation. In: European Conference on Visual Media Production, CVMP 2005, pp. 92–101 (2005)
13. Billinghurst, M., Campbell, S., Chinthammit, W., Hendrickson, D., Poupyrev, I., Takahashi, K., Kato, H.: Magic book: Exploring transitions in collaborative ar interfaces. SIGGRAPH 2000 (2000)
14. Zhang, Z.: A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 1330–1334 (2000)
15. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge University Press, Cambridge, UK (2004)
16. Pharr, M., Fernando, R.: GPU Gems 2: Programming Techniques For High-Performance Graphics And General-Purpose Computation. Addison-Wesley Professional, Reading (2005)
17. Kalva, H., Christodoulou, L., Mayron, L., Marques, O., Furht, B.: Challenges and opportunities in video coding for 3D TV. In: IEEE International Conference on Multimedia & Expo (ICME), Canada, pp. 1689–1692 (2006)