A Stereoscopic Movie Player with Real-Time Content Adaptation to the Display Geometry

Sylvain Duchêne, Martin Lambers, Frédéric Devernay

A stereoscopic movie player with real-time content adaptation to the display geometry

Sylvain Duchêne[a], Martin Lambers[b],Frédéric Devernay*[a]
[a]PRIMA team, INRIA Grenoble Rhône-Alpes, France
[b]Computer Graphics and Multimedia Systems Group, University of Siegen, Germany

ABSTRACT

3D shape perception in a stereoscopic movie depends on several depth cues, including stereopsis. For a given content, the depth perceived from stereopsis highly depends on the camera setup as well as on the display size and distance. This can lead to disturbing depth distortions such as the cardboard effect or the puppet theater effect. As more and more stereoscopic 3D content is produced in 3D (feature movies, documentaries, sports broadcasts), a key point is to get the same 3D experience on any display. For this purpose, perceived depth distortions can be resolved by performing view synthesis. We propose a real time implementation of a stereoscopic player based on the open-source software Bino, which is able to adapt a stereoscopic movie to any display, based on user-provided camera and display parameters.

Keywords: Stereoscopy, 3D video, depth perception, depth-preserving disparity mapping, view synthesis, real time, stereoscopic player

1. INTRODUCTION

Many media are now available in stereoscopic 3D without offering the best viewing condition due to the depth distortion that appear on different displays due to the variety of screen sizes and distances. However this distortion may be reduced through view synthesis methods, which usually involve three steps[1]: computing the stereo disparity, applying a disparity-dependent mapping to the original views and compositing the resulting images. In this paper, we do not focus on the first step: while stereo disparity computation is still an active research topic, state-of-the art methods give results that are satisfying for our application.

In the first part of this paper, we focus on the main constraints on the camera setup used to shoot a stereoscopic movie for a given display configuration. Then, we discuss the choice of our view synthesis method through a view synthesis model built on shooting and viewing geometries. We make the assumption that the movie is correctly rectified: a pixel from the left image corresponds to a pixel on the same horizontal line in the right image, i.e. there is no vertical disparity or vertical misalignment. In the second part, we present our pipeline and an efficient way to implement it on the GPU using several render passes satisfying video frame rate performance and high quality. Finally, we discuss our results, the problems that may appear, such as visual artifacts, and how to solve them.

Our main research goals and contributions are to propose a stereoscopic movie player performing real-time content adaptation to the display geometry with a framework using both CPU and GPU.

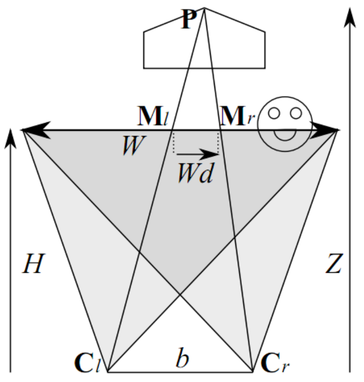
2. SHOOTING CONSTRAINTS DESCRIPTION AND CONTENT ADAPTATION

Projecting a stereoscopic movie on different screen sizes and distances will produce different perceptions of depth, which implies that a movie is shot for a particular display configuration. If a stereoscopic movie is viewed without modification, three main issues have to be considered: eye divergence, image scaling and roundness factor[2].

Figure 1 shows the various geometric parameters describing the camera and display configurations.

*frederic.devernay@inria.fr; phone +33 4 76 61 52 58; www.inrialpes.fr

This work was done within the 3DLive project, supported by the French Ministry of Industry www.3dlive-project.com



Symbol	Camera	Display
C_l, C_r	camera optical center	eye optical center
P	physical scene point	perceived 3-D point
M_l, M_r	image points of P	screen points
b	camera interocular	eye interocular
H	convergence distance	screen distance
W	width of convergence plane	screen size
Z	real depth	perceived depth
d	left-to right disparity (as a fraction of W)	

Figure 1. Shooting and viewing geometries can be described using the same small set of parameters.

Since triangles M_lPM_r and C_lPC_r are homothetic, the disparity d can be written as a function of the real depth Z :

$$d = \frac{b}{W}\frac{Z-H}{Z}, \text{ or } Z = \frac{H}{1-\frac{dW}{b}} \quad (1)$$

2.1 Eye divergence

The perceived depth can be expressed by the same way since the horizontal disparity in the images and the screen disparity respectively expressed as a fraction of image width and fraction of screen width are equal:$d = d'$. This results in:

$$Z' = \frac{H'}{1-\frac{dW'}{b'}} \qquad (2)$$

By eliminating the disparity$d$, from (1) and (2), we obtain:

$$Z' = \frac{H'}{1-\frac{W'}{b'}\left(\frac{b}{W}\frac{Z-H}{Z}\right)} \quad (3)$$

Eye divergence occurs when $Z' < 0$, i.e.$d' > bW'$; object placed at infinity in the real scene$(Z \to \infty)$ will cause eye divergence if and only if $W'/b' > W/b$.

2.2 Scale ratio

The image scale ratio σ' represents how much an object placed at depth Z seems to be enlarged (eg. $\sigma' > 1$) or reduced (eg. $\sigma' < 1$) in $X$and Ydirection with respect to objects present in the convergence plane $s = \frac{H}{Z}$.

$$\sigma' = \frac{s'}{s} = \frac{H'}{Z'}\frac{H}{Z} = \frac{1-dW'/b'}{1-dW/b} (4)$$

On-screen objects get a scale ratio equal to 1 since the disparity is 0. Moreover the relation between Z and Z' is non linear except if $\frac{W'}{b'} = \frac{W}{b}$.

2.3 The roundness factor

The roundness factor ρ measures how much the object proportions are affected for an object of dimensions $\delta X, \delta Z$ in the width and the depth directions at a depth Z, perceived as an object of dimensions $\delta X', \delta Z'$ at depth Z'.

$$\rho = \frac{\delta Z'}{\delta Z}\Big/\frac{\delta X'}{\delta X} = \frac{\delta Z'}{\delta Z}\Big/\frac{W'/s'}{W/s} = \sigma'\frac{W}{W'}\frac{\delta Z'}{\delta Z} \quad (5)$$

The roundness factor of an object on the screen plane ($Z = H$ and $Z' = H'$)is:

$$\rho_{screen} = \frac{W}{W'}\frac{\delta Z'}{\delta Z} = \frac{b}{H}\frac{H'}{b'} \quad (6)$$

Note that despite common belief, the on-screen roundness factor does not depend on the screen size, but on the screen distance. This on-screen roundness factor is equal to 1 if and only if$b'/b = H'/H$.

In order to get perfect shape reproduction, the roundness factor should be equal to 1 everywhere. This implies that the geometric parameters have to satisfy $b'/b = W'/W = H'/H$. Consequently, the only camera configurations that preservethe roundness factor everywhere are scaled versions of the viewing geometry[2].

2.4 View synthesis model

A generalized depth preserving view synthesis model built on shooting and viewing geometries parameters which satisfies both described constraints can be written as follows[3].

A pixel in the left image which has coordinates(x_l, y) and disparity$d_l$maps to the following pixel in the interpolated view:

$$(x_c + (x_l + wd_l - x_c)\sigma''(d_l) + w(d''(d_l) - d_l), y_c + (y - y_c)\,\sigma''(d_l)), \qquad (7)$$

where w is the image width in pixels, d'' is the mapping from the original disparity to the synthesized disparity:

$$d''(d) = \frac{H'b}{(HW' - H'W)d + H'b} \qquad (8)$$

and σ'' is the disparity-dependent image scaling:

$$\sigma''(d) = \frac{H'b}{(HW' - H'W)d + H'b}. \ (9)$$

Similarly, a pixel in the right image which has coordinates(x_r, y) and disparity$d_r$maps to the following pixel in the interpolated view:

$$(x_c + (x_r - x_c)\sigma''(d_r) + w(d''(d_r) - d_r), yc + (y - y_c)\,\sigma''(d_r)) \ (10)$$

Note that σ'' and d'' depend on the parameters that describe the shooting and viewing geometries.

As we only need to synthesize one view to maximize the quality[3], we prefer a simpler depth-preserving disparity mapping synthesis method over view synthesis method. *Depth-preserving disparity mapping* preserves the depth proportions with a roundness factor equal to 1 for on-screen object, and it avoids eye-divergence, although the disparity-dependent image scaling σ'' is not applied, resulting in off-screen objects having a roundness different than 1.

Expressions (7) and (10) become:

$$(x_l + w * d''(d_l), y) \qquad (11)$$

and

$$(x_r + w * (d''(d_r) - d_r), y), \qquad (12)$$

and the pixel-dependent blending factors to be applied to the two warped images are:

$$\alpha_i^l = |d_i^r|/(|d_i^l| + |d_i^r|)\ , \alpha_i^r = 1 - \alpha_i^l \qquad (13)$$

The computation of the disparity d''at each pixel can be easily parallelized on a GPU architecture, since it is a closed-form computation that only depend on values at that pixel.

3. RENDERING PIPELINE OVERVIEW

Most view synthesis algorithms require a stereo pair and the associated disparity map(s), but rendering a movie at video frame rate imposes some constraints on the performance required. Our approach can be divided into a sequence of steps:

1. We make sure than the stereo input video sources are rectified. This ensures that each pixel in the left image corresponds to a pixel on the same horizontal line in the right image.
2. We compute the disparity maps from this stereo sequence using a coarse-to-fine method. This method can be implemented on the GPU[4] and run almost at video-rate on today's GPUs. Then we encode the disparity map in a video stream using the luminance channel.

3. Both the disparity and the video streams are sent to the stereoscopic movie player. The CPU decodes each stream and the two decoded image pairs (left and right images, left-to-right and right-to-left disparity maps) are sent to the GPU.
4. Two views are rendered from the stereo pair displaced using vertex buffer grid geometry, and depth discontinuities are handled by using transparency.
5. We then composite these views to obtain a stereo pair adapted to the viewing conditions and sent to the display using a standard stereoscopic display stream format (Top/bottom, Above/Below, Checkerboard…)
6. An optional pass can be inserted before conversion to the stereoscopic format, in order to remove artifacts using a confidence map[3].

Figure 2 describes graphically the data flow from the original images and disparity maps to the synthesized image pair.

3.1 Implementation framework

To implement our real-time solution we used the framework provided by the open-source stereoscopic movie player Bino[5] which already implements many required features in a unified framework, for example multithreaded and synchronized video stream decoding. It also supports different stereoscopic display formats, such as top/bottom, above/below, and checkerboard ,and offers a linear OpenGL pipeline. The decoding is processed through the open-source library FFmpeg, which supports a large number of codecs as well as multi-threaded decoding on the CPU.

Moreover, the framework takes care of using linear RGB pixel values over the pipeline, which is essential during all the process[6]. Intermediate color buffers may lose precision in lower value ranges when stored as 8-bit linear images. For this reason, either 8-bit sRGB images or 16-bit linear-RGB images are used as intermediate buffers.

Our disparity map was produced with a coarse-to fine dense stereo matching algorithm, which is suitable for real-time performance[4] to be as close as possible to an embedded solution in a stereo movie player. Both left and right disparity maps are encoded in a gray scale video. All the videos were encoded using the H.264 codec for its quality, performance and popularity as it's widely used by many streaming Internet sources. As a disparity can be either positive or negative, an encoding range needs to be chosen to define the 0 disparity.

3.2 Asymmetric precomputation pass

The first rendering pass converts the disparity video input stream by applying the correct scale and offset, and performs the computation of the texture element displacement. Since recent OpenGL versions allow the use of multiple render targets, left and right streams can be processed on the same pass sharing the same GLSL shader using two framebuffers to store the result. This pass does not use any diverging branches, which makes it very fast. xd_i^l and xd_i^r represent the displacement from a point x from L to I and from R to I.

Input Textures: left and right disparity maps
Input parameters: viewing and shooting geometries parameters

$$xd_i^l = w * d''(d_l) \qquad xd_i^r = w * (d''(d_r) - d_r)$$

$$\alpha_i^l = |d_i^r| / (|d_i^l| + |d_i^r|) \qquad \alpha_i^r = 1 - \alpha_i^l$$

Output_buffer [0]: xLI, $d''(d_l)$, d_l, α_i^l
Output_ buffer [1]: xRI, $d''(d_r) - d_r)$, d_r, α_i^r

Note that the current pixel position isn't applied yet.

All the computation is done in the fragment shader, since the vertex shader only needs to transfer the texture coordinates.

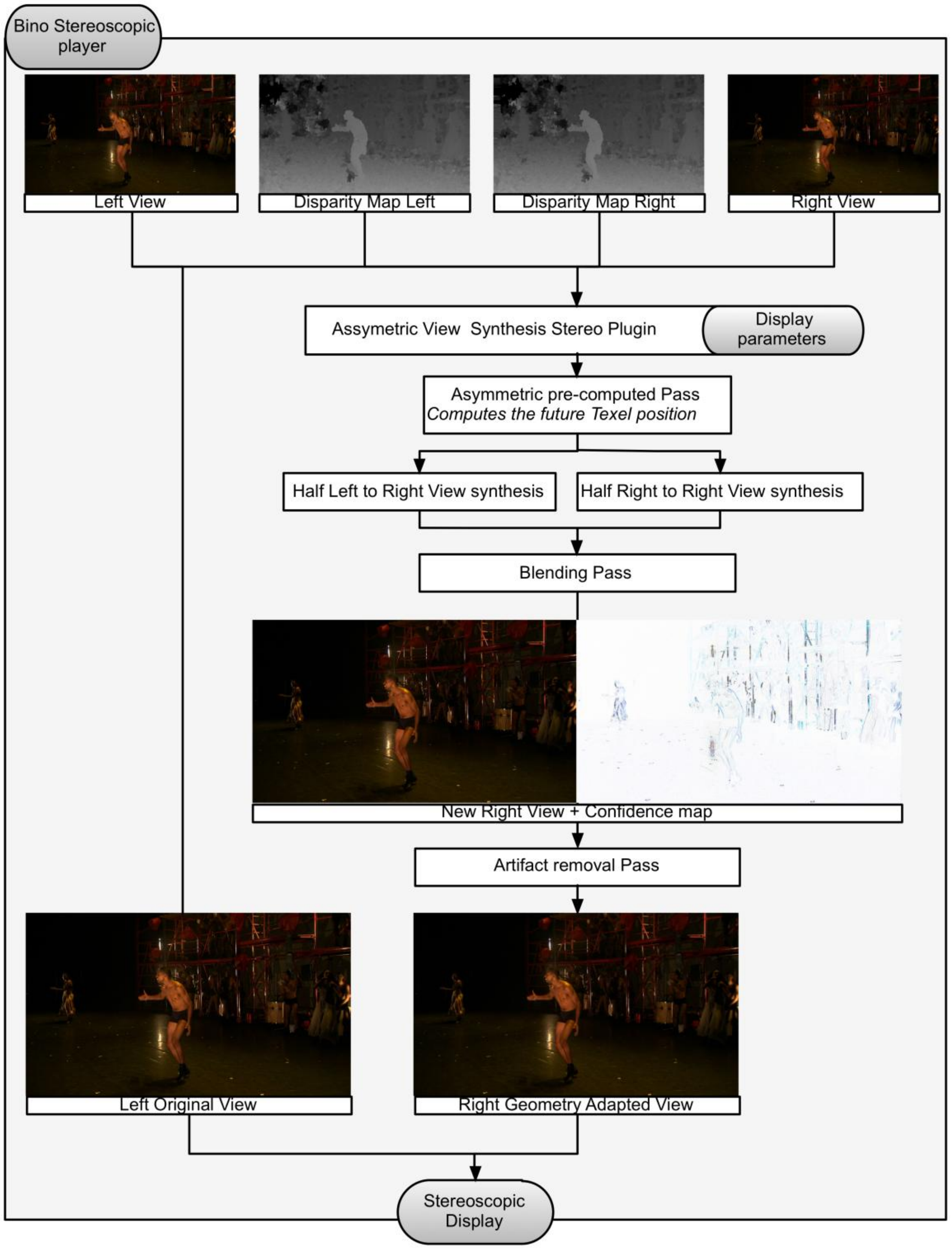

Bino Stereoscopic player
Left View
Disparity Map Left
Disparity Map Right
Right View
Assymetric View Synthesis Stereo Plugin
Display parameters
Asymmetric pre-computed Pass
Computes the future Texel position
Half Left to Right View synthesis
Half Right to Right View synthesis
Blending Pass
New Right View + Confidence map
Artifact removal Pass
Left Original View
Right Geometry Adapted View
Stereoscopic Display

Figure 2. Rendering pipeline overview: the two views and the two disparity maps are uploaded as textures on the GPU, and all subsequent processing is performed on the GPU.

3.3 Rendering half views

Image based rendering can be performed with explicit geometry or with implicit geometry. Our use case is closer to view interpolation than to free viewpoint methods, as our goal is to adapt the content using disparity mapping[7], which can be seen as a depth-dependent baseline interpolation method.

Using a grid model where each vertex is displaced by the OpengGL pipeline rasterizer makes sense, since we can also handle occlusions that way, by using the disparity itself as the z-component and using a Z-buffer. We preferred using standard OpenGL shaders rather than Cuda or OpenCL in the pipeline, because switching the GPU between the OpenGL and the Cuda/OpenCL modes takes too much time on consumer-level graphics cards. By using only one large indexed vertex buffer object grid in HD (1920*1080) made from a single triangle strip, we can render both view in two passes. The grid is initialized with texture coordinates corresponding to an identity transform.

Input Textures: assymetric_framebuffer ($xd_i^l, d''(d_l), d_l, \alpha_i^l$)
Input parameters: threshold (1.0)

$$x_i^l = gl_{Vertex}.x + xd_i^l$$

$$gl_{Position} = gl_{ModelViewProjectionMatrix} * (x_i^l, \quad gl_{Vertex}.y, -d''(d_l), 1.0)$$

The vertex shader applies the displacement on the x-axis for each vertex from the grid using the mapping from the previous pass but also on the z-axis using only the synthesized disparity to handle self-occlusions of the displaced grid.
Detecting depth discontinuities is easy as the previous buffer can be sampled in its neighborhood: when $\|xd_i^l - x_{-1}d_i^l\|$>threshold, the blending factors has to be set to 0.0. By default the threshold has to be 1.0 to detect any elongated triangles in the displaced grid.

Output [0]: $\mathrm{RGBvaluesfromleftview}, \ \mathrm{blendingfactor}$

The fragment shader just applies the color from the texture through the textures coordinates attached to each displaced vertex. Note that left and right borders must be refilled using the last candidate with a blending factor set to 0.0.
Figure 3 shows the intermediate result with the two rendered half-views.

3.4 Blending

The two synthesized views are blended together using the asymmetric disparity mapping blending factors, combined with the discontinuity-detection blending factors.

Input : left and right contribution to synthesize the new view

$$\alpha_i^l = |d_i^r|/(|d_i^l| + |d_i^r|)\alpha_i^r = 1 - \alpha_i^l$$

Output: compose views using blending factors

4. RESULTS

The resulting software plays videos fluently at Full HD resolution (1080p25) on a quad-core 2.8GHz with a GeForce GTX480 GPU. The overall quality is good, but some visual artifacts may appear because of errors present in the disparity maps. Many of these artifacts can actually be detected by image processing[8], and we are working on GPU-based artifact removal methods that could be used for real-time rendering.

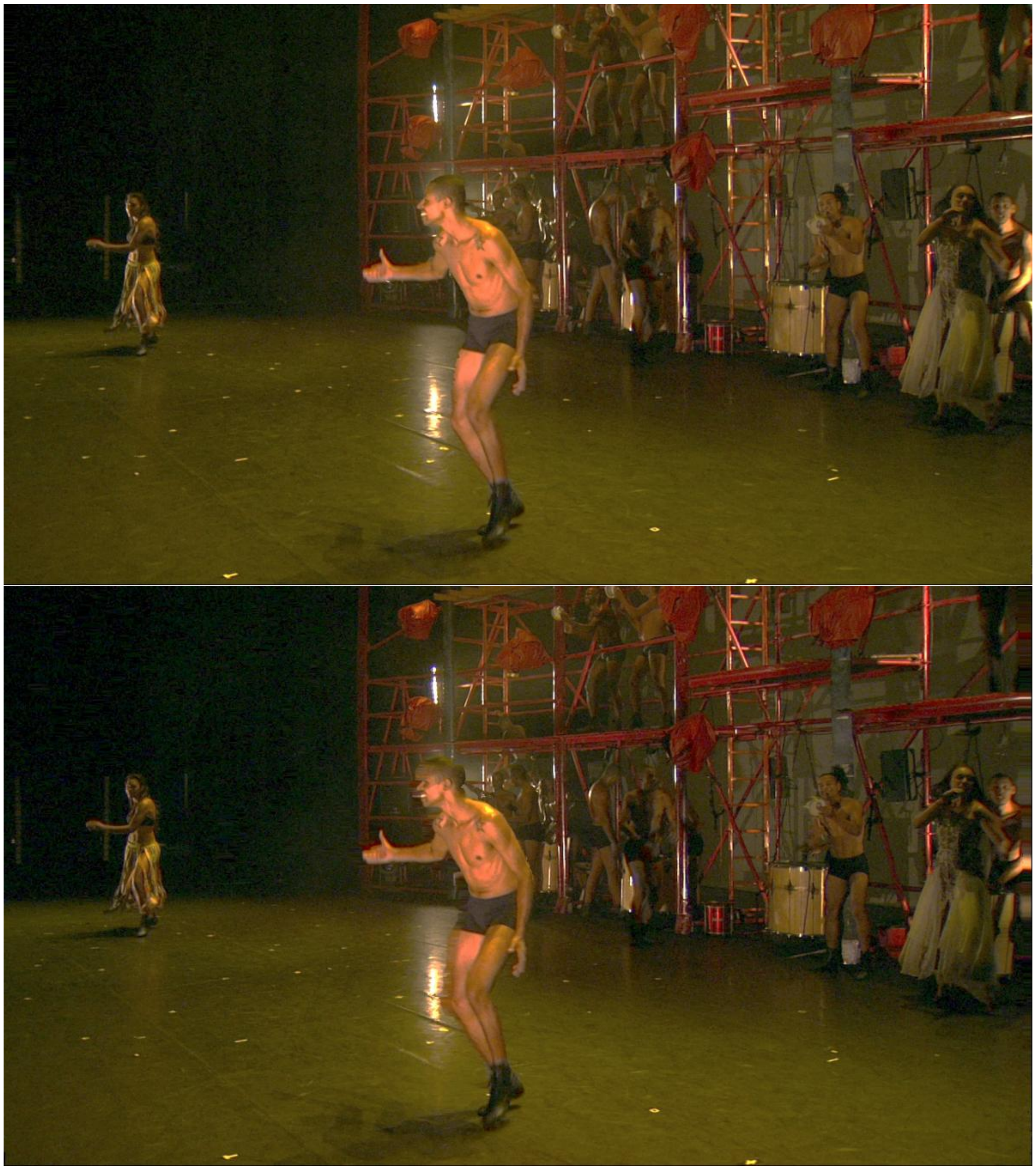


Figure 3. Left view mapped to the synthesized view (top), right view mapped to the synthesized view (bottom).

5. CONCLUSION AND FUTURE WORK

We presented a real-time stereoscopic movie player with content adaptation to the display geometry through a framework that unifies video streams decoding on the CPU and depth preserving view synthesis into a set of chained algorithmic building blocks on the GPU. Future work should be focus on real-time artifact removal methods, and on the computation and transmission of the disparity maps.

The disparity maps should be computed on the fly from the left and right video streams, and can either be transmitted together with the left and right videos, or could be computed on the client side (for example on a set-top box).

Another subject of interest is to detect which shots can be left untouched (so that artifacts caused by view synthesis can be avoided completely), based on the amount of divergence and on the global amount of depth distortions caused by viewing the unmodified shot.

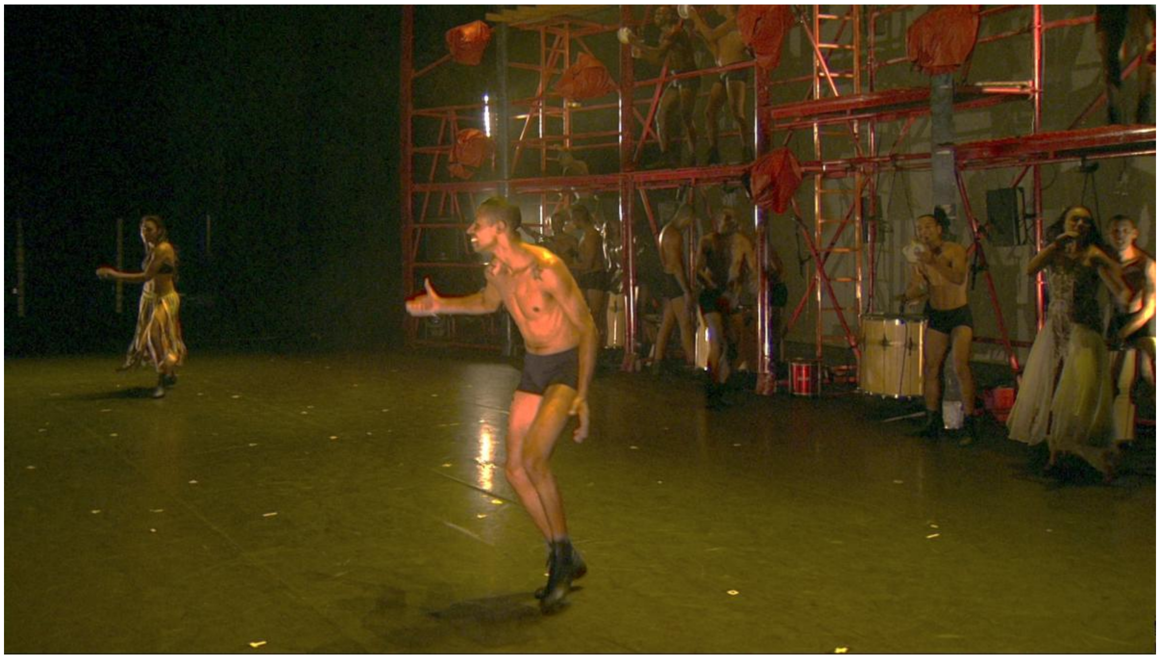


Figure 4. New composed view with depth-preserving synthesis method.

REFERENCES

[1] Sammy Rogmans, Jiangbo Lu, Philippe Bekaert, and Gauthier Lafruit,“Real-time stereo-based view synthesis algorithms: A unified framework and evaluation on commodity GPUs,” Signal Processing: Image Communication, 24(1-2):49–64 (2009). ISSN 0923-5965. doi: 10.1016/j.image.2008.10.005. Special issue on advances in three-dimensional television and video.

[2] Frédéric Devernay, Sylvain Duchêne, and Adrian Ramos-Peon,“Adapting stereoscopic movies to the viewing conditions using depth-preserving and artifact-free novel view synthesis,” Stereoscopic Displays and Applications XXII, volume 7863 (2011). SPIE. doi: 10.1117/12.872883.

[3] Frédéric Devernay and Sylvain Duchêne, “New view synthesis for stereo cinema by hybrid disparity remapping,”Proc. International Conference on Image Processing (ICIP), pages 5–8, Hong Kong, (2010). doi: 10.1109/ICIP.2010.5649194.

[4] M. Sizintsev, S. Kuthirummaly, S. Samarasekeray, R. Kumary, H. S. Sawhneyy, and A. Chaudhryy,“GPU accellerated realtime stereo for augmented reality,” Proc. Intl. Symp. 3D Data Processing, Visualization and Transmission (3DPVT), 2010.

[5] Martin Lambers, “Bino: free 3D video player”http://bino3d.org/ Accessed: december 2011

[6] Larry Gritz and Eugene d'Eon,“The Importance of Being Linear,” GPU Gems 3, Chapter 24, Hubert Nguyen (ed.), Addison-Wesley (2007), ISBN 0321515269

[7] Manuel Lang, Alexander Hornung, Oliver Wang, Steven Poulakos, Aljoscha Smolic, and Markus Gross, “Nonlinear disparity mapping for stereoscopic 3D,” ACM SIGGRAPH 2010 papers, SIGGRAPH ’10, pages 75:1–75:10(2010). ISBN 978-1-4503-0210-4. doi: 10.1145/1833349.1778812.

[8] Frédéric Devernay and Adrian Ramos-Peon, “Novel view synthesis for stereoscopic cinema: detecting and removing artifacts,” Proc.1st international workshop on 3D video processing, 3DVP ’10, pp. 25–30 (2010). ACM. ISBN 978-1-4503- 0159-6. doi: 10.1145/1877791.1877798.