Multiple View Generation for Auto-stereoscopic Displays

Stephan Beck, Mathias Schneider, Bernd Fröhlich

Virtual Reality Systems Group
Fakultät Medien, Bauhaus-Universität Weimar
Bauhausstrasse 11
99423 Weimar
contact email: stephan.beck@uni-weimar.de

Abstract: This paper presents several approaches for accelerating the view generation for auto-stereoscopic displays by exploiting the capabilities of current graphics cards. Besides rendering the scene multiple times, we also investigate 3D image warping for generating intermediate views from a number of reference images. While this approach may introduce artifacts, the particular view arrangement of a multi-view auto-stereoscopic display allows us to directly detect and fill holes during warping at a small additional cost. We show that multi-pass rendering and 3D image warping benefit from the utilization of the geometry shader and perform up to 40 percent faster compared to a multi-pass approach without geometry shader.

Keywords: auto-stereoscopic displays, geometry shader, image-based rendering, post-rendering warping

1 Introduction

Recently, stereoscopic displays are getting more and more popular and are on their way to the consumer market. Auto-stereoscopic displays promise to be the ultimate solution – especially if they support more than two views. Rendering multiple views of a 3D scene can be achieved by using the standard real-time graphics pipeline in a naive multi-pass fashion. This results in a linearly increasing rendering time with respect to the number of views. However, there is a lot of coherence between a number of closely spaced views. Rendering algorithms exploiting this coherence should lead to a slower than linear growth of rendering times – at least for a limited number of views.

In this paper we investigate several approaches for accelerating the view generation by exploiting capabilities of recent graphics cards. In particular, the unified shader model has the benefit that the available processing units can be freely assigned to the vertex, geometry and fragment shaders to allow for a better balancing between these different pipeline stages. In particular, the geometry stage which is represented by the geometry shader unit scales better than in the past. Our idea is to use the geometry shader to generate multiple geometry streams for different views from a single geometry input stream.

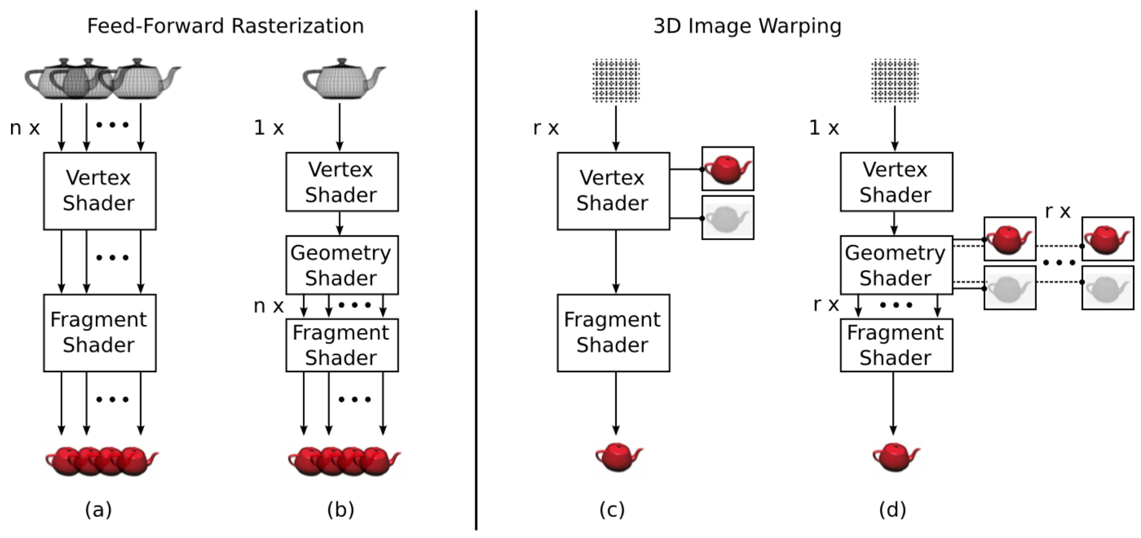

Figure 1: Investigated rendering pipelines: (a) naive multi-pass rendering, (b) layered rendering using the geometry shader, (c) r reference views are warped within the vertex shader to generate one view from r reference views, (d) using the geometry shader one view can be generated from r reference views in a single pass.

Figure 1 gives an overview of various multi-view rendering approaches. Our results show that a geometry-shader-based multi-view implementation generally performs faster than simply using multiple render passes. We assume that the better cache coherence is the main reason for this improvement. Depth-image warping is an alternative to directly generating multiple views at the expense of potentially introducing artifacts. In case of multi-view rendering for auto-stereoscopic displays these artifacts appear mostly along image lines and can be often reduced by a simple hole-filling strategy. Our new algorithm directly detects and fills holes during the warping process executed in the geometry shader unit at only a small additional cost.

2 Multiple View Generation

Various techniques for auto-stereoscopic displays exist [KPG+07]. In this research the focus is on parallax barrier displays such as the MV-24AD3 from Newsight GmbH [New10]. In the MV-24AD3 currently eight perspective views are combined in a spatially multiplexed image based on sub-pixels. Latest auto-stereoscopic displays provide up to 64 views [Ger10].

The views that are presented on an auto-stereoscopic display have some common characteristics which can be considered for development: All viewpoints have the same distance to their neighbor viewpoint. For a given point in one view, the point lies on the same y-coordinate in all other views - if not occluded by an object.

Creating the spatially multiplexed image for an auto-stereoscopic display is done in two steps: First, the necessary n views have to be generated. Second, these views have to be combined to a single multiplexed image which is specially parameterized for the output display [Gra10].

There are two basic methods for generating n views of a scene: The standard graphics pipeline and 3D image warping.

Using the standard graphics pipeline, the scene is rendered n times from slightly different viewpoints. Modern graphics cards support stereoscopic vision already i.e. 3D Vision from Nvidia [Nvi10], but generating more than two views for one scene is not supported. The straightforward method therefore is to use multiple render passes, see also figure 1 (a). We refer to this as baseline approach. Shader Model 4.0 introduces the geometry shader as a new shader unit which is localized between the vertex and the fragment shader. Compared to the vertex shader, the geometry shader is able to modify and generate entire primitives such as points, lines or triangles [PBSN07]. Another OpenGL extension, which is highly coupled with the geometry shader, is `EXT_tex- ture_array` [Bro08]. A texture array is a collection of one- or two-dimensional textures ordered in layers. The access of an array of two-dimensional textures is similar to a three-dimensional layer where the depth value describes the layer [PBSN07].

The combination of these two extensions facilitates layered rendering which is shown in figure 2 schematically. By passing the model-view-projection matrices for each view to the geometry shader each primitive can be transformed correctly and routed to the corresponding layer in the texture array. Using this technique multiple views can be rendered in a single pass and the geometry has to be fetched only once from the GPU memory.

The second method to generate multiple views is to only render a single reference view and then use 3D image warping [MMB97][Bec09] to create the remaining views by re-projecting the color and depth buffer of a reference view into a new perspective. This technique is known as depth-image-based rendering (DIBR). Because 3D image warping is independent from the scene complexity it is faster for complex scenes, while the quality of the output is lower since missing scene information leads to artifacts in the final image [Bec09].

Both techniques, feed-forward rasterization and 3D image warping can be combined: For example every second view of the n views can be rendered and the missing views can be generated by warping. This rendering pipeline is depicted in Figure 1 (c) and (d). Note that the reference views can be generated either in a multi-pass fashion (MP) or using the geometry shader (GS).

3 Related Work

Sorbier et al. [FdSB08] proposed to use the geometry shader to duplicate and transform every triangle of the scene to another viewport already. In their implementation they route the single views to the output textures using a multiple rendering target (MRT) within the fragment shader. Since this solution has only one depth buffer for all views available, the renderer is using the Painters Algorithm. Due to this restriction, rendering complex scenes is not feasible by their algorithm.

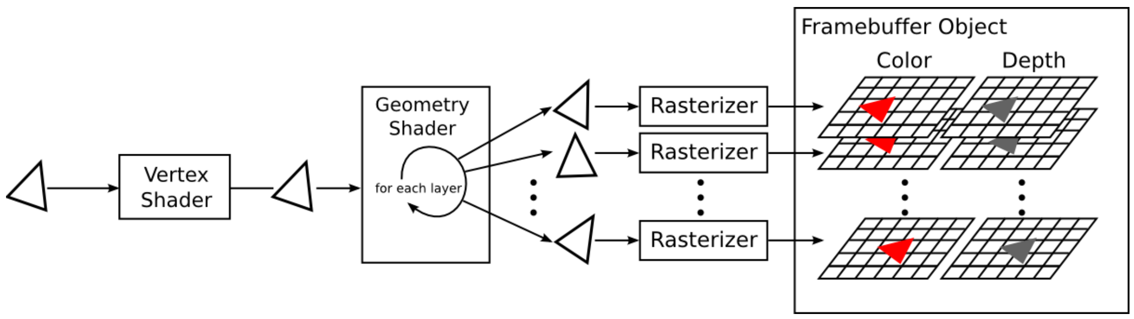

Figure 2: Layered rendering: An incoming triangle is duplicated and transformed within the geometry shader and then rasterized into the layers of a texture array (color and depth) that is attached to a framebuffer object.

Marbach et al. [FdSB08] overcome this drawback by combining geometry shaders and layered rendering. The single views are rendered to the layers of a texture array while another texture array is used as depth buffer.

Kwan-Jung et al. [OYH09] introduce a hole-filling method that uses the available depth information in the neighborhood. Instead of filling a hole by interpolating the colors at the edges of the hole, the filling is done based on the depth value. Not only the color of the neighbor pixels but also their depth is considered during filling. It is assumed, that the background is occluded by the object in the foreground and therefore the color of the background area should be used rather than the color of the foreground. While their work focuses on 3D video, we suggest a similar solution to that problem in the context of real-time rendering.

4 Implementation of Rendering Pipelines

In order to allow a flexible setup of the discussed view generation techniques, we developed a software framework to abstract the work flow as described in subsection 2.1 by combining different implementations of *producers* (rendering views), *warpers* (warping views out of the produced views) and *viewers* (multiplexing and displaying the results for the output device) to a rendering pipeline. Figure 4 gives an overview of the framework architecture.

We implemented a producer similar to [Mar09]. The test scene is described by a flat scene graph, using the scene graph implementation of gloost [WB10]. The modelview and the modelview-projection-matrix are precomputed on the CPU and passed to the geometry shader. This avoids a matrix-multiplication for each view within the geometry shader on the GPU. The output primitives are then routed to the corresponding layer in the texture array by the `gl_Layer` command (see listing in figure 5) and phong-shaded within the fragment shader using a single point light.

In addition, we implemented a warper using a geometry shader and a warper using a vertex shader as a reference, see also figure 1. Similar to the producer described above

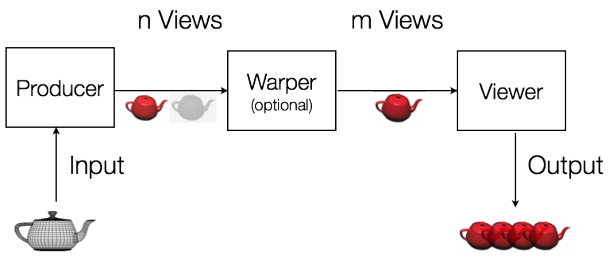

Figure 3: Framework architecture.

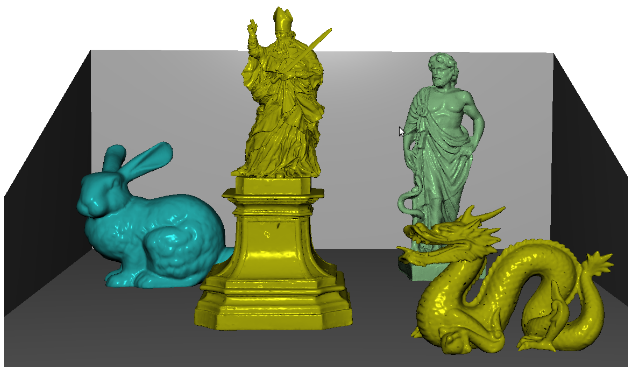

Figure 4: Test scene consisting of 1.2 M triangles.

the warping of several reference views can be done in one render pass instead of having a render pass for each view. Therefore the geometry shader duplicates the vertices of the proxy geometry for each reference view on the fly. This results in less transfer cost from the GPU memory because the proxy geometry needs to be passed to the pipeline only once.

4.1 Results and Discussion

In a first test setup we compared our geometry shader rendering pipeline as described in the previous section to the naive multi-pass implementation, see also figure 1 (a) and (b) respectively. The resolution for each view was 1920x1080 and the scene consisted of five objects with 1.2 M triangles in total and is stored in display lists for fast object rendering. The test system was equipped with a Intel Core i7 CPU 940 2.93GHz and a Nvidia GeForce GTX 480. The render times for several view configurations are shown in figure 6.

While [Mar09] found the results of layered rendering slower, current GPUs seem to be faster and can improve the rendering performance compared to a multi-pass approach by 35 percent for 8 views and up to 40 percent for 4 views.

We think that the reason for this is the new architecture of the Nvidia GPUs. Our geometry shader producer needs to fetch the geometry only once from the GPU memory and all subsequent computations for the necessary views can be done on the same geometry. Note that this results in a coherent data processing of the whole geometry set. When generating more than 12 views the stage after the geometry shader becomes a bottleneck, e.g. the routing into the layers of the texture array, and the performance drops compared

```
uniform int nLayers;
uniform mat4 modViewProjMatrix[8]; // 8 Matrices, one for each view
void main(){
 // for each view (layer in texture array)
 for(int layer = 0; layer < nLayers; ++layer){
   // for each vertex of the incoming triangle
   for(int i = 0; i < 3; ++i){
    // apply the material
    gl_FrontColor = gl_FrontColorIn[i];
    gl_TexCoord[0] = gl_TexCoordIn[i][0];
    // ...apply other attributes like vertex normal

    // transform it into the current view
    gl_Position =   modViewProjMatrix[layer]
                       * gl_PositionIn[i];
    // emit the vertex of the triangle
    EmitVertex();
   }
   // set output layer within texture array
   gl_Layer = layer;
   // create the triangle
   EndPrimitive();
 }
}
```

Figure 5: Simplified (OpenGL/GLSL-) source code for the geometry-shader producer.

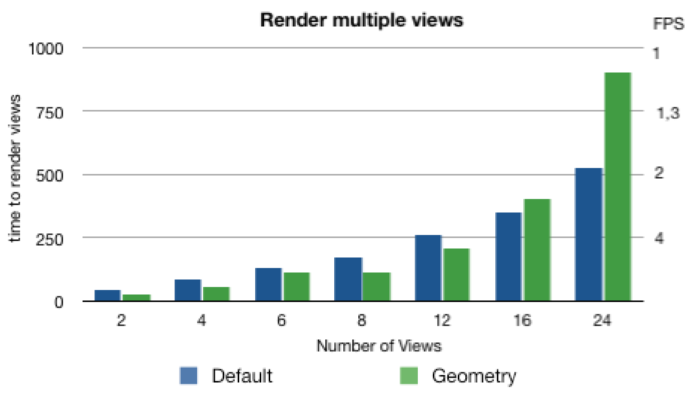

Figure 6: Render time in ms to generate up to 24 views for a scene with 1.2 M triangles. Default: multi-pass producer, Geometry: geometry-shader producer.

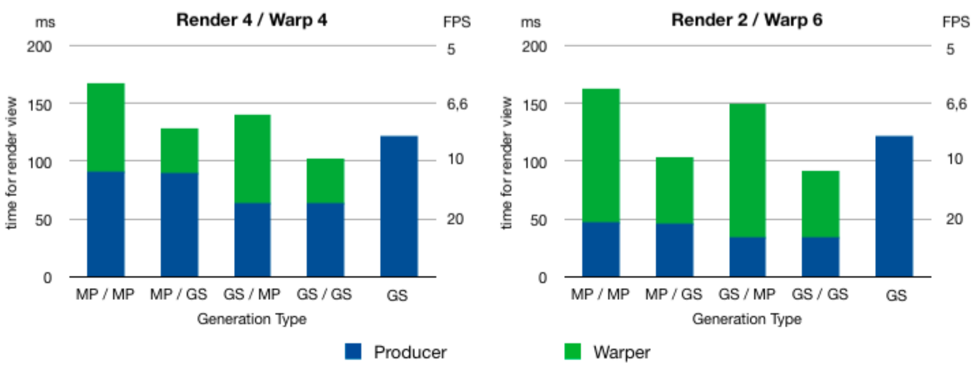

Figure 7: Time in ms for generating eight views of a scene with 1.2 M triangles. GS: geometry-shader producer, MP: multi-pass producer.

to the multi-pass version.

In another test-setup we combined the two different implementations - multi-pass (MP) and geometry shader (GS) - for the producer and the warper and measured the time for generating eight views. In the first case, four views are produced and four are warped. In the second case, the outer two views are produced and the six remaining views are warped. Each warped image is synthesized from two produced images. The results are shown in figure 7. In both cases the combination of a geometry-shader producer and a geometry-shader warper is the fastest configuration to create the eight views. Compared to the layered rendering pipeline (rightmost bar in the diagram) the performance improvement is about 15 percent (Render 4/ Warp 4) and 25 percent (Render 2/ Warp 6).

Because the geometry shader is able to duplicate primitives and therefore can sample more pixels within a reference view, the resolution of the proxy geometry does not have to be the same as the resolution of the reference view. We therefore modified our geometry-shader warper to use a reduced proxy-geometry resolution and measured the rendering time to warp a single view for several configurations. As shown in figure 8 the warping performance can be improved up to 50 percent by reducing the vertex load. The reason for this speedup is that less vertices need to be fetched from GPU memory. This coincides with our previous comparison of multi-pass warping vs. geometry-shader warping.

5 Hole Filling

As it can be seen in the previous results, the best render performance can be achieved by rendering only some reference views and use warping to generate the remaining views. In the following we describe a method that detects and fills holes during warping in the geometry shader directly.

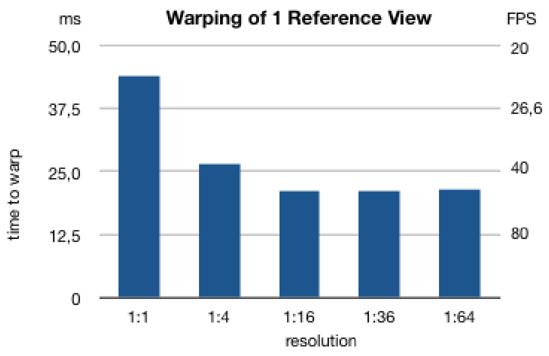

Figure 8: Time to warp one reference view using reduced proxy grid resolutions, e.g. 1:4 means that a single geometry shader kernel samples 4 pixels within the reference view.

5.1 Algorithm

The warper uses a point grid as proxy geometry where each vertex corresponds to a pixel in the reference view. The basic idea is to detect holes during warping within the geometry shader and close them immediately in the same pass. Therefore two neighboring points of the reference image are warped and their horizontal distance in screen space is calculated. If it is below a certain threshold a line segment between these two points is created and sent to the rasterizer. Thereby, small holes (often caused by rounding errors) can be filled.

If the gap between two neighboring points is too large, the background surface is extended in the direction to a virtual point v to fill the hole. Figure 9 shows such a case. Between the current pixel c and the next pixel n a hole of one pixel appears. Since c is a point in the background, its corresponding surface is extended in the direction from the previous point p to the current point c. The virtual point v is constructed as shown by equation 1 and 2 where λ_1 and λ_2 are the unknown scale factors for the direction vector $\vec{d}$ (defined by $(\vec{p} - \vec{c})$) and the position $\vec{n}$.

$$\vec{v} = \vec{p} + \lambda_1 \cdot (\vec{p} - \vec{c}) = \lambda_2 \cdot \vec{n} \tag{1}$$

$$\begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} \cdot \lambda_1 - \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \cdot \lambda_2 = - \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{2}$$

Because of the special nature of auto-stereoscopic displays, the y-values for all vectors are the same. Thus, the linear system of equations can be solved for the unknown λ_2 as in equation 3.

$$\lambda_2 = \frac{d_x \cdot p_z - d_z \cdot p_x}{d_x \cdot n_z - d_z \cdot n_x} \tag{3}$$

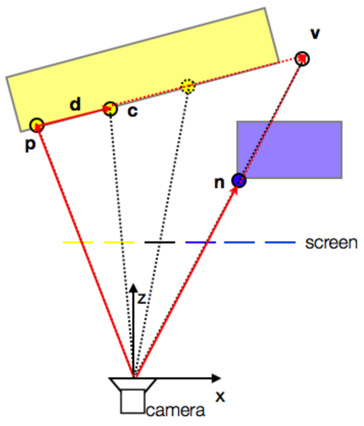

Figure 9: Construction of the virtual point v along the direction d in the geometry shader. Therefore the hole (black pixel in the screen) is filled with the background color (yellow).

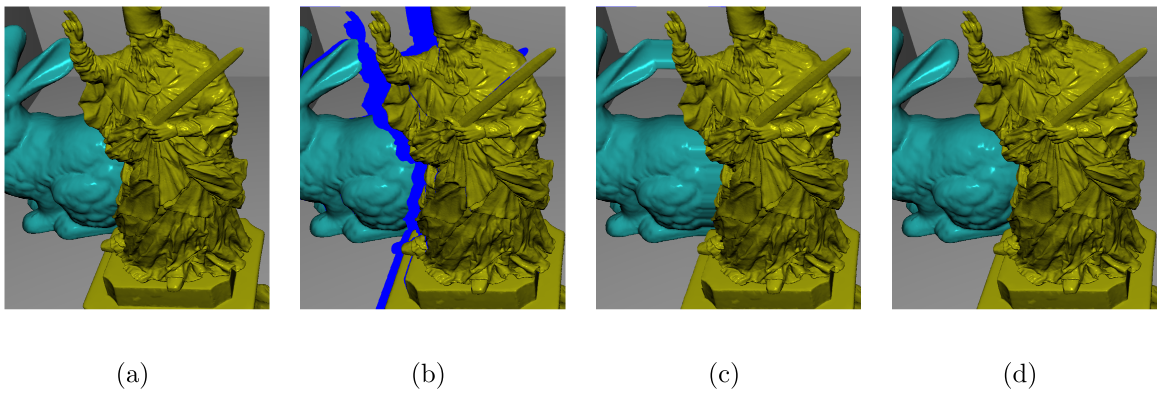

Figure 10: Warping results for complex backgrounds: (a) Reference image, (b) warping of (a) without hole filling, (c) warping of (a) with our hole-filling method, (d) correct rendered image for comparison.

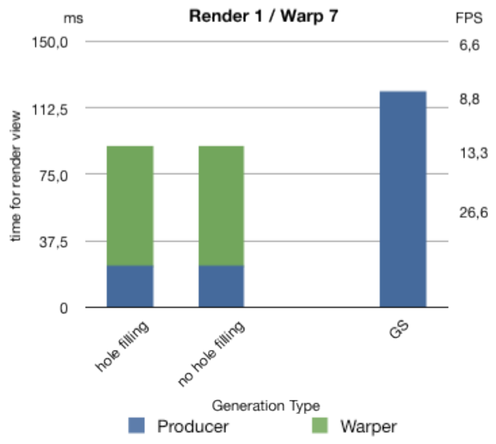

Figure 11: Time in ms for warping 8 views of a scene with 1.2 M triangles using hole filling.

5.2 Results and Discussion

Figure 10 shows the results with and without the presented hole-filling algorithm. Compared to the rendered view (see figure 10(d)) there are not all holes filled correctly. Because the color of the new virtual point is the same as the current point, the algorithm can only produce good results in case of a homogeneous background. If the nature of the background has a high frequency, clearly visible lines appear which results in poor visual quality. Therefore complex surfaces can not be extended correctly. Another issue is that our algorithm always assumes that the surface is extended behind the occluding object which is not always correct (e.g. see the incorrectly extended ear of the Stanford bunny in figure 10(c).).

As shown in figure 11, our algorithm requires only little additional time compared to a warper implementation without hole filling and it can still reduce the render time in our test scenario by 25 percent (blue reference bar).

6 Conclusions and Future Work

Our results show that the use of geometry shaders on current GPUs accelerates multi-view rendering compared to a multi-pass implementation by up to 40 percent for generating 4 views and up to 35 percent for generating 8 views. In addition geometry shaders can also speed up the warping of several reference views into a final image by replicating the proxy geometry on the fly. Our experiments demonstrate an improvement of about 50 percent.

Our hole-filling method can reliably fill small holes of the size of a few pixels with only a small additional rendering cost in the geometry shader. The filling quality strongly depends on the nature of the scene and its depth complexity. The method produces good results in case of homogeneous backgrounds. Complex background surfaces cannot be extended correctly with our simple approach, the geometry shader might be used to generate additional vertices where it is necessary and drop vertices where the surface is homogeneous. However, without having access to the actual scene geometry it is impossible to completely overcome the problem of incorrectly filled holes. In image processing there exist many approaches, which might be usable to further improve the current algorithm, i.e. edge smoothing [SMD+08] or texture synthesis [Kom06].

Using the geometry shader to generate multiple geometry streams from a single input stream may accelerate other image generation processes as well. We will investigate if and when the geometry-shader rendering pipeline can improve the performance of regular stereo rendering where only two views have to be generated. In particular advanced shading models that incorporate textures within the fragment stage should benefit from the increased coherency. Furthermore, we would like to investigate how the geometry shader can improve the generation of multiple shadow maps in a single pass. Even the generation of a sequence of images in an animation may be accelerated by generating the images for multiple time steps in a single pass.

References

[Bec09] Stephan Beck. Untersuchungen zur verbesserung der bildwiederholrate durch hardware-unterstütztes tiefenbild-warping, 2009. Diplomarbeit, Bauhaus-Universität Weimar.

[Bro08] Pat Brown. Ext_texture_array. Extension description: `http://developer.download.nvidia.com/opengl/specs/GL_EXT_texture_array.txt`, April 2008.

[FdSB08] Vincent Nozick Francois de Sorbier and Venceslas Biri. Gpu rendering for autostereoscopic displays. In *4th International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT'08)*. ACM, June 2008.

[Ger10] Televisions.com Germany. Singaporean manufacturer presents autostereoscopic displays. Sunny Ocean: `http://www.televisions.com/tv-news/Singaporean-manufacturer-presents-autostereoscopic-displays.php`, February 2010.

[Gra10] Armin Grasnick. Universal 4d multiplexing of layered disparity image sequences for pixel and voxel based display devices. In *Three-Dimensional Image Processing (3DIP) and Applications*, San Jose, CA, USA, 2010. IS&T / SPIE.

[Kom06] Nikos Komodakis. Image completion using global optimization. In *CVPR '06: Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 442–452, Washington, DC, USA, 2006. IEEE Computer Society.

[KPG+07] Robert L. Kooima, Tom Peterka, Javier I. Girado, Jinghua Ge, Daniel J. Sandin, and Thomas A. DeFanti. A gpu sub-pixel algorithm for autostereoscopic virtual reality. *Virtual Reality Conference, IEEE*, pages 131–137, 2007.

[Mar09] Jonathan Marbach. Gpu acceleration of stereoscopic and multi-view rendering for virtual reality applications. In *VRST '09: Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, pages 103–110, New York, NY, USA, 2009. ACM.

[MMB97] William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. In *I3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 7–16, New York, NY, USA, 1997. ACM.

[New10] Newsight. Our technology. Product Homepage: `http://www.newsight.com/index.php?id=81`, May 2010.

[Nvi10] Nvidia. 3d vision. Product Homepage: `http://www.nvidia.com/object/3D_Vision_Main.html`, May 2010.

[OYH09] Kwan-Jung Oh, Sehoon Yea, and Yo-Sung Ho. Hole filling method using depth based in-painting for view synthesis in free viewpoint television and 3-d video. In *PCS'09: Proceedings of the 27th conference on Picture Coding Symposium*, pages 233–236, Piscataway, NJ, USA, 2009. IEEE Press.

[PBSN07] Suryakant Patidar, Shiben Bhattacharjee, Jag Mohan Singh, and P. J. Narayanan. Exploiting the shader model 4.0 architecture. Technical report, International Institute of Information Technology, Hyderabad, March 2007.

[SMD$^+$08] Aljoscha Smolic, Karsten Müller, Kristina Dix, Philipp Merkle, Peter Kauff, and Thomas Wiegand. Intermediate view interpolation based on multiview video plus depth for advanced 3d video systems. In *Proceedings of 15th International Conference on Image Processing (ICIP 2008)*, pages 2448–2451, San Diego, CA, USA, 2008. IEEE International Conference on Image Processing.

[WB10] Felix Weiszig and Stephan Beck. gloost. Project Homepage: `http://developer.berlios.de/projects/gloost`, May 2010.