A UNIFIED GRAPHICS RENDERING PIPELINE FOR AUTOSTEREOSCOPIC RENDERING

Aravind Kalaiah[1], *Tolga K. Capin*[2]

[1]NVIDIA Inc.
[2]Bilkent University

ABSTRACT

Autostereoscopic displays require rendering a scene from multiple viewpoints. The architecture of current-generation graphics processors are still grounded in the historic evolution of monoscopic rendering. In this paper, we present a novel programmable rendering pipeline that renders to multiple viewpoints in a single pass. Our approach leverages on the computational and memory fetch coherence of rendering to multiple viewpoints to achieve significant speedup. We present an emulation of the principles of our pipeline using the current-generation GPUs and present a quantitative estimate of the benefits of our approach. We make a case for the new rendering pipeline by demonstrating its benefits for a range of applications such as autostereoscopic rendering and for shadow map computation for a scene with multiple light sources.

Index Terms— Computer Graphics, 3D Graphics, autostereoscopic rendering.

1. INTRODUCTION

Computer graphics has traditionally focused on rendering to one viewpoint at a time. Consequently, most of the rendering algorithms and speciality graphics hardware are geared towards handling Single Viewpoint Rendering (SVR). While this approach addresses the core applications of the graphics industry, we are witnessing a rapid growth in alternate scenarios that require rendering a scene from multiple viewpoints. A good example of the application of Multiple Viewpoint Rendering (MVR) are the autostereoscopic displays [Hal97]. Autostereoscopic displays are a generalization of the conventional displays by their ability to emit different light in different directions. This allows them to convey stereo parallax to multiple viewers at the same time [SS99, MP04, SMG05]. Recent technological advances have brought several commercial products to the market place (e.g. 42-inch Philips 3D6C01 display, Sharp Corporation's LL-151-3D monitor and Actius AL3D laptop, and NTT DoCoMo SH505i mobile phone).

Current graphics standards and rendering hardware provide mechanisms for rendering to multiple viewpoints. However, this is mainly a basic system that invokes the traditional SVR rendering system multiple times: once for each viewpoint. This is a correct but very costly solution since it ignores the inherent coherence in the computation and memory access of the rendering process. Currently, the only way to do such a concurrent rendering is to use multiple-GPU solutions such as NVIDIA's SLI architecture.

In this paper we propose modifications to the OpenGL graphics pipeline which allows rendering to multiple viewpoints in a single pass. Our approach is to make MVR a generalization (as opposed to a specialization) of SVR. Our approach leverages on the coherence of computation and memory access of MVR and leads to faster rendering and conserves other resources such as bus bandwidth and battery power. While current GPUs cannot be configured to directly implement our new pipeline, we emulate our approach indirectly on the GPU.

2. RELATED WORK

2.1. Stereoscopic Rendering

Multiple viewpoint rendering is a generalization of stereoscopic rendering. Traditionally, stereoscopic rendering involves processing separately the left and right eye. This is still the model in use in graphics APIs such as OpenGL [SA04]. Adelson et al. [ABC91] were the first to walk through the rendering pipeline and discuss how the x-axis coherence in device coordinates can be used for simultaneously rendering a triangle to both images.

Adelson and Hodges [AH93] accelerate stereoscopic raytracing by warping a raytraced left image to the right eye view and raytracing the right image only for a subset of the pixels. He and Kaufman [HK96] accelerate stereoscopic volume rendering by re-projecting the samples made while ray casting for the left view to the right image plane. Fu et al. [FBP96] compute the left image of a polygonal scene and compute the right image by warping it. The holes in the right image are filled by interpolation. Wan et al. [MZQK04] accelerate stereoscopic rendering of opaque volumes by computing the left image and warping it to the right image and filling the leftover gaps of the right image by raycasting. Fehn [Feh04] generates multiple views from a single image with an accompanying depth buffer using 3D warping. Fehn handles the hole filling problem by smoothing the depth buffer with a Gaussian filter.

2.2. Multiple Viewpoint Rendering

We refer to the process of rendering a scene from several arbitrary viewpoints as multiple viewpoint rendering. Halle [Hal98] discusses a method for rendering to an autostereoscopic display from a spatio-perspective volume representation after it is constructed from the epipolar images of the scene. His method can be extended to rendering to arbitrary viewpoints. Govindarju et al. [GLY03] use two different Graphics Processing Units (GPUs) for visibility processing from the eye point and the light position. The multiple render target feature of OpenGL 2.0 [SA04] can be used to render to multiple buffers at the same time. However, since all the buffers share the same rasterization process, it cannot be used for MVR. Nvidia's dual-GPU SLI combination can accelerate rendering by designating each GPU to compute a different frame. Our approach can do this with a one frame delay at a much lower hardware cost.

2.3. Autostereoscopic Displays

Autostereoscopic displays have a rich history. We refer the reader to Sexton and Surman [SS99], Matusik and Pfister [MP04] and Sandin et al. [SMG05] for a good overview of the technology behind autostereoscopic displays.

Halle [Hal97] discusses the guidelines to display 3D content for autostereoscopic displays. Yoshida et al. [YMH99] present ways to configure the viewing parameters for a better control of the perceived depth. Jones et al. [JLHE01] discuss the mathematics behind setting up the virtual camera for a given viewer position relative to the autostereoscopic display.

3. UNIFIED RENDERING PIPELINE

We refer to our new rendering pipeline as the unified pipeline since the task of rendering to the different viewpoints is done in a single pass (see figure 1). In our model, the GPU pipeline begins with the vertex shader receiving the vertices with attributes. The Vertex Shading Unit has both viewpoint-independent and viewpoint-specific global registers to hold the uniform variables. The viewpoint-independent registers can carry information such as the surface material and the light sources while the viewpoint-specific registers can hold information such as the projection matrix, the viewport matrix, and the eye position. The Vertex Shaders can also allow per-vertex view dependent information such as the reflection vector. The vertex shaders then output both viewpoint-independent information such as diffuse lighting and viewpoint-specific information such as post-projection vertex positions through the appropriate registers. This model allows the vertex shader to compute complex view-independent computation such as precomputed radiate transfer only once and reuse the results for all the viewpoints. Subsequently, after all the vertices of a primitive have been processed by the vertex shader, it is assembled together into a primitive and is replicated into multiple primitives, one for each viewpoint, by the Primitive Replication Unit. This adds a unique tag, called the view tag, to the primitive which identifies the viewpoint that the primitive is destined for and it also ensures that a replicated primitive only carries the information that is relevant for its designated viewpoint. Each of the replicated primitives is then clipped to the frustum corresponding to its viewpoint generating multiple primitives in the process if necessary. These primitives can then be processed by the traditional rasterizer which outputs pixels with both interpolated viewpoint-specific and viewpoint-dependent parameters. The rasterizer also adds a view tag to each output pixel to help identify the pixel buffer that the pixel is destined for. The pixels can then be processed by the traditional pixel processing model. The Pixel Shading Unit can be preloaded with both viewpoint-specific and viewpoint-independent information. Pixel processing tests such as the alpha test, the depth test and the stencil test will require a separate buffer for each viewpoint and a separate framebuffer (or render target) has to be maintained for each viewpoint as well. After a pixel is processed by the pixel shaders, and if it passes all the pixel tests, it is written to the framebuffer (or render target) designated to its viewpoint. Our approach seamlessly upgrades the current rendering pipeline and should fit well with architectural enhancements such as the Unified Shaders [Dog05] and Geometry Shaders [Bly06].

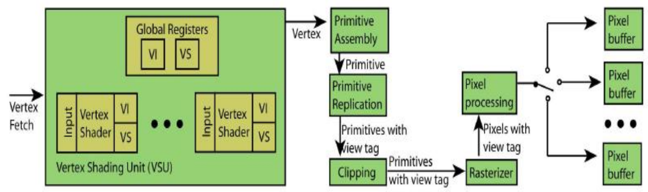

Figure 1: *The Unified Rendering pipeline for multi-view autostereoscopic rendering. Here VI and VS refer to viewpoint-independent and viewpoint-specific registers respectively.*

Our approach ensures that the most common computation and memory fetch requirements for the different viewpoints are done only once. In particular, barring viewpoint-specific information, the data traffic going into the GPU is still the same as a pipeline for monoscopic rendering. Unlike before, the vertex shader does not have to redo viewpoint-specific computation and does not have to fetch the texels multiple times. Similarly the pixel shaders make better use of the texture cache since the pixels of the replicated primitives are likely to have similar texture fetch patterns. All these serve to reduce computation, power consumption, memory fetch, and the bus bandwidth consumption of autostereoscopic graphics applications.

We note here that in the pipeline described above, the number of viewpoints supported by the pipeline can be different from the number of viewpoints required for stereoscopic rendering. For example, if the display device requires rendering from six views and the pipeline supports rendering to at most fours viewpoints, a multi-pass scheme can be used which renders to four viewpoints in one pass and to the remaining two viewpoints in the next pass. Also, since the viewpoints can be any arbitrary viewpoints, it can be used for other purposes such as rendering for the rear view mirror in games and for shadow computation from multiple light sources.

4. EMULATING THE UNIFIED PIPELINE

Our new pipeline described in section 3 serves as the guideline for future GPU implementations for autostereoscopic rendering. In this section we emulate the unified rendering pipeline using current GPUs. Our emulation serves to estimate the performance gains to be obtained from the unified rendering pipeline.

Our emulation pipeline is illustrated in figure 2. It follows the principles of the unified rendering pipeline by rearranging the order of the operations. The emulation pipeline addresses all these essential issues: 1) pixel buffer replication, 2) primitive replication, 3) vertex processing, 4) clipping, and 5) interlacing. Pixel buffer replication refers to assigning a separate pixel buffer for each viewpoint. We handle this by rendering to an offscreen buffer that is partitioned according to the viewpoint configuration. Alternately, if the Multiple Render Target (MRT) feature [SA04] is available in the GPU, separate buffers can be directly assigned to individual viewpoints.

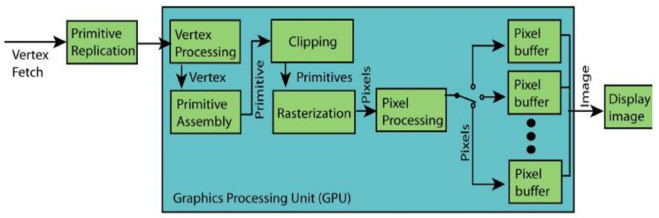

Figure 2: *The modified rendering pipeline for Stereoscopic*

Primitive replication is not directly supported by the current generation of GPUs without geometry shaders. We emulate this indirectly by using the vertex array capability of OpenGL. We explain our approach for the case of triangle replication using the `glDrawArrays` function call. A similar approach can be employed for replicating other primitives. In our approach we preload V dummy triangles in the graphics memory as a primitive array (V denotes the number of viewpoints). We pack the vertex position of each of the three vertices of the triangle into unused attribute slots of the vertex such as the texture coordinate slot. For rendering a triangle, instead of rendering it by making three `glVertex` calls, we render multiple triangles by making a call to `glDrawArrays` to render the V dummy triangles. This has the effect of sending nearly as much information to the GPU as the monoscopic case, avoiding the traditional approach of sending the information V times. This approach to primitive replication can also be implemented using similar OpenGL functions functions such as `glDrawElements`.

When the replicated vertices arrive at the pixel shader for the vertex processing we need to assign each incoming vertex of the array to act as one of the original vertex being rendered to a unique viewpoint. In other words, each incoming vertex from the array needs to be assigned a unique (triangle vertex, viewpoint) combination. For this we assign the (x, y)- coordinate values of a vertex v_i of a dummy rectangle t_j to be (i, j). When a dummy vertex arrives at the vertex shader, its (x, y, z)-coordinate is set to the appropriate vertex attribute that holds the original vertex coordinate.

After a vertex has been assigned its (x, y, z)-coordinate value it has to be transformed and the projected to its designated viewpoint. For this we multiply it by the common modelview matrix and the viewpoint-specific matrix. This has be to be followed by a viewpoint-specific viewport transformation. Since the OpenGL library does not expose viewport transformation to the vertex shader we use an indirect approach to viewport transformation, by computing the same transformations on vertex shaders.

5. RESULTS

Rendering to autostereoscopic displays requires careful placement of the virtual cameras. The actual configuration of the virtual cameras in terms of their numbers, relative position, and their view frustum depends on the physical properties of the display system, the depth range of the scene, and the desired perceived depth. In this paper we work with a symmetric grid configuration (see figure 3(a)) but our approach is extensible to any arbitrary configuration.

We compute the camera properties using the techniques proposed by Jones et al. [JLHE01]. We maintain a common modelview matrix for the scene. We also maintain a viewpoint-specific matrix that will induce a translation to the eye of the viewpoint from the common eye followed by a projection according to that viewpoint's camera parameters. Rendering by the conventional approach would involve rendering the scene to an offscreen- or a back-buffer for each of the viewpoints and combining the images together in a final pass by interlacing. For a two-view display with horizontal parallax, the interlacing would involve mixing the columns of the two images such that the final image would have odd-column pixels from the left image and the evencolumn pixels from the right image. In our case, turning on the autostereoscopic feature of the display of our hardware reduced the horizontal resolution by half. Hence a renderer can work with half the half-resolution for the offscreen/back buffers.

We implemented our work on a Sharp Actius AL3D laptop with a two-view parallax-barrier autostereoscopic display. We consider the following viewpoint configurations: 1 viewpoint (1×1), 2 viewpoints (2×1), 3 viewpoints (3×1), 4 viewpoints (2×2), 6 viewpoints (3×2), and 9 viewpoints (9×9). We render our test cases to a fixed 800×600 offscreen buffer (p-buffer). This means higher number of viewpoints decrease the resolution of individual viewpoints.

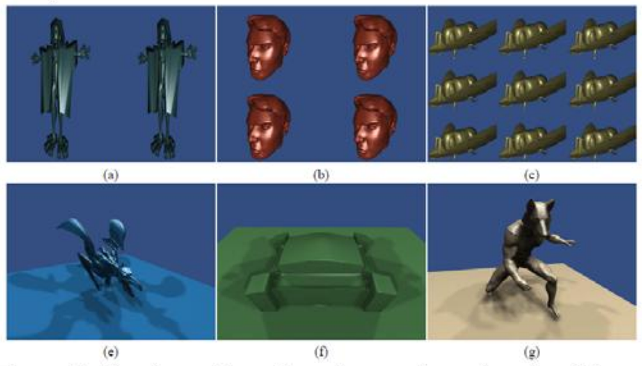

Figure 3: *The viewpoint configurations and experiment models used for our experiment. (a) Alien (3280 tris.), (b) Head (11556 tris.), (c) Bee (37420 tris.), (d) Gargoyle (40348 tris.), (e) Atrium (27542 tris.), (f) Werewolf (133536 tris.).*

We detail our results in table 1. We measure the speed of rendering in terms of the time taken to render to the offscreen buffer (reported in frames per second (FPS)). As the table shows, our rendering speed is slower than conventional rendering when only one viewpoint is rendered to. However, when the number of view points becomes 2 we see a sudden jump to a positive gain. Thenafter the gain continues to rise in the positive territory. All but one of the test cases were at least 50% better than the conventional approach for a 9 viewpoint configuration. We found no consistent relationship between the number of polygons and the speedup in rendering except that the gains of the unified pipeline apply to models of all sizes. For small models, the time of rendering did not differ significantly for small viewpoint numbers but as the number of viewpoints increased it began to make an impact on their speed. Amongst the larger models, some had more speedup than others. We attribute this to the coherence in their triangle ordering.

Table 1: *Comparison of FPS speeds of conventional autostereoscopic rendering (C) with proposed emulation (U)*

View	Alien		Head		Atrium		Bee		Gargoyle		Werewolf	
	C	U	C	U	C	U	C	U	C	U	C	U
1×1	1338.6	1002.1	352.9	319.2	197.8	140.0	121.3	100.8	125.2	94.6	35.2	26.6
2×1	752.8	989.3	186.2	224.3	100.2	123.7	62.1	93.7	63.6	89.9	17.7	26.1
3×1	522.7	969.2	125.7	170.8	67.1	87.3	41.8	65.6	42.7	61.4	11.8	19.9
2×2	383.6	956.2	94.9	137.9	50.5	66.9	31.5	50.7	32.1	47.2	8.8	15.0
3×2	272.3	543.6	63.7	99.8	33.7	47.2	21.1	34.7	21.5	32.2	5.9	10.1
3×3	184.5	295.2	42.6	70.6	22.5	32.3	14.1	23.6	14.3	21.8	3.9	6.8

CONCLUSIONS

We presented a new rendering pipeline for autostereoscopic displays which hinges on the principle of primitive replication. Our new pipeline seamlessly upgrades the existing pipeline and can be used for other applications such as shadow computation from multiple light sources that require multiple viewpoint rendering. We presented an emulation of our pipeline using current GPUs. Our results show that the emulation pipeline outscores the conventional approach by a large margin and shows much promise for a hardware implementation of our pipeline.

For future work we would like to point out that the OpenGL API is inadequate for multiple viewpoint rendering. Ideally, the application should simply specify the multiple viewpoints once and send the geometry information to the GPU only once while the GPU should generate the images with this information. Our unified pipeline handles this model and the OpenGL API should be modified to handle such a model. Our emulation approach can be used for several multiple viewpoint rendering use cases such as rendering large LoD datasets, shadow computation, and precomputing future frames during interactive navigation.

Acknowledgments. This work was partially supported by EC FP6 3DTV Project (Grant no: FP6-511568).

REFERENCES

[ABC_91] ADELSON S.J., BENTLEY J.B., CHONG I.S., HODGES L.F., WINOGRAD J.: *Simultaneous Generation of Stereoscopic Views*. Tech. Rep. GIT-GVU-91-07, 1991.

[AH93] ADELSON S., HODGES L.: Stereoscopic ray tracing. *The Visual Computer 10*, 3 (1993), 127–144.

[Bly06] Blythe, D. 2006. The Direct3D 10 system, SIGGRAPH '06. ACM Press, New York, NY, 724-734.

[Dog05] DOGGETT M.: Xenos: XBOX360 GPU. In *EUROGRAPHICS, New console architectures* (Sept. 2005).

[FBP96] FU S., BAO H., PENG Q.: Accelerated rendering algorithm for stereoscopic display. *Computers and Graphics 20*, 2 (1996), 223–229.

[Feh04] FEHN C.: Depth-image-based rendering (DIBR), compression, and transmission for a new approach on 3DTV. In *Proc. SPIE Vol. 5291,* 2004, pp. 93–104.

[GLY03] GOVINDARAJU N. K., LLOYD B., YOON S., SUD A., MANOCHA D.: Interactive shadow generation in complex environments. *ACM Trans on Graphics 22*, 3 (2003), 501–510.

[Hal97] HALLE M.: Autostereoscopic displays and computer graphics. *SIGGRAPH Comput. Graph. 31*, 2 (1997), 58–62.

[Hal98] HALLE M.: Multiple viewpoint rendering. In *SIGGRAPH Conference Proceedings* (1998), pp. 243–254.

[HK96] HE T., KAUFMAN A.: Fast stereo volume rendering. In *IEEE Visualization* (1996), pp. 49–57.

[JLHE01] JONES G. R., et al. .: Controlling perceived depth in stereoscopic images. In *Proc. SPIE Vol. 4297, p. 42-53,* 2001.

[MP04] MATUSIK W., PFISTER H.: 3D TV: A Scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. *SIGGRAPH 2004,* pp. 814–824.

[MZQK04] M.WAN, ZHANG N., QU H., KAUFMAN A. E.: Interactive stereoscopic rendering of volumetric environments. *IEEE Trans. on Vis. and Comp. Graphics 10*, 1 (2004), 15–28.

[SA04] SEGAL M., AKELEY K.: The OpenGL Graphics System: A Specification (Version 2.0), 2004.

[SMG_05] SANDIN D. J., et al. The varrier autostereoscopic virtual reality display. *Proc. SIGGRAPH 2005,* 894–903.

[SS99] SEXTON I., SURMAN P.: Stereoscopic and autostereoscopic display systems. In *IEEE Signal Processing Magazine* (May 1999), pp. 85–99.

[YMH99] YOSHIDA S., et al. : A technique for precise depth representation in stereoscopic display. In *Computer Graphics International* (June 1999), pp. 80–84.