Guided High-Quality Rendering

Thorsten Roth[1,2], Martin Weier[1,3], Jens Maiero[1,2],
André Hinkenjann[1], Yongmin Li[2]

[1]Institute of Visual Computing, Sankt Augustin, Germany
[2]Brunel University London, United Kingdom
[3]Saarland University Computer Graphics Lab, Saarbrücken, Germany

Abstract. We present a system which allows for guiding the image quality in global illumination (GI) methods by user-specified regions of interest (ROIs). This is done with either a tracked interaction device or a mouse-based method, making it possible to create a visualization with varying convergence rates throughout one image towards a GI solution. To achieve this, we introduce a scheduling approach based on Sparse Matrix Compression (SMC) for efficient generation and distribution of rendering tasks on the GPU that allows for altering the sampling density over the image plane. Moreover, we present a prototypical approach for filtering the newly, possibly sparse samples to a final image. Finally, we show how large-scale display systems can benefit from rendering with ROIs.

1 Introduction

Photorealistic image synthesis largely depends on the complex, physically-based process of GI. The underlying rendering equation [1] is often solved by employing Monte Carlo methods, which in turn depend on random variables. Basically, these variables are used to create rays, which are traced through the scene and reflected at their intersection points with the scene geometry according to material and illumination properties. Computing GI with MC methods poses two strongly related problems due to its random nature: Noise and convergence behaviour. At first, MC approaches converge quickly, but with an $\mathcal{O}(1/\sqrt{n})$ asymptotic error, this convergence slows down quickly.

As a consequence, cutting the noise for n samples in half requires the computation of $4n$ samples. This is a serious issue when physically-based GI is used in performance-critical applications in which quality is crucial for judging certain properties of the rendered image, which is the case in areas like design review or architectural visualization. Also, this statement can certainly be extended to games. However, determining the number of samples required to achieve a specific quality without performing the actual rendering process is generally not possible. Modern large display setups like 4k projections allow for detailed visualizations, while modern PCs allow for the parallel implementation of the aforementioned rendering algorithms on GPUs. While a single GPU can handle a 4k setup memory-wise, even when a large number of buffers is required

(e.g., for post-processing), its performance is still not high enough to perform real-time MC-based GI at such resolutions. We present an approach for coping with today's situation, where designers would like to have a quick, high-quality visualization of products at high resolutions, while modern GPUs are not yet fast enough to compute high-resolution, physically-based GI in an acceptable time frame. Our basic assumption is that an image can be divided into areas with different significance for the user. Consequently, the focus will mostly be put on regions with higher significance, making the convergence of outer areas less important as they are not perceived in a detailed way by the human visual system. Nevertheless, outer areas are still important in order to provide a visual context. Our main contributions are:

1. A general framework for adaptive rendering based on a user-centered context
2. Two interaction metaphors for choosing the relevant image regions
3. A scheduling method based on *Sparse Matrix Compression (SMC)* for efficiently rendering an image with a varying sample density on the GPU
4. A basic filtering approach providing an improved visual context

The GI rendering method we employ throughout this paper is Path Tracing. However, the presented methods are applicable to all rendering methods dealing with similar issues. All benchmarks and other results are based on a resolution of 3840×2160 pixels, unless stated otherwise. Our work focuses mainly on large projection systems. Lacking a 4k projection system in our lab, we use our large, high-resolution display wall HORNET, consisting of 7×5 displays with a 1080p resolution each, to mimic such a system.

2 Related Work

Over the last decades several visualization algorithms that make use of focus and context techniques and ROIs have been published. Most of them try to support the user in understanding complex data in 2D and 3D, e.g., the work in [2] for reading documents or [3] for graphs and hierarchies. Another focus and context technique was introduced in [4] to display giga-pixel images on a large, high resolution display wall. For Giga-pixel images , dealing with a system's limited bandwidth is the major challenge. To overcome this issue, ROIs that can be specified by the user to guide which data to load, defined by a specific position and resolution are employed. Focus and context techniques for 3D data sets are often used in combination with direct volume rendering as the sampling rate for individual rays can be altered at run time. Volume rendering with focus and context can be categorized into (a) distortion lenses, which use approaches similar to the Document Lens, altering the sampling with lenses like ROIs [5], (b) cutaway illustrations, which allow exploring the data using cut planes or x-ray vision, as proposed in [6] and (c) multi-resolution focus and context approaches like the system in [7], where a multi-resolution technique based on eye-tracking is introduced, altering the resolution and sampling rates of volume rendering based on the actual viewing direction. These systems use ROIs specified by the

user's gaze in combination with a linear falloff of the visual acuity in the visual peripheral (foveated rendering). They either use eye-tracking or only the head-tracked view-direction. User-centered approaches like [8] use the latter method to generate constant frame rates in virtual environments, combining different rendering approaches. Content-based models of visual attention are used in [9]. This approach also includes a user-centered GI method for prerendered animations. Another approach that utilizes foveated rendering was introduced in [10], employing raster graphics to create three layers with different sampling rates that are blended to a final image. In addition, they use different geometric levels of detail employing tesselation of parametric surfaces to further reduce the computational workload. Geometric simplification based on ROIs and gaze has been proposed in [11–13]. The system proposed in [11] uses ray tracing in combination with gaze-based geometric simplification with multi-resolution meshes. Weier et al. [13] present a simplification based on a hybrid voxel representation. Another method that is based on ray tracing without geometric simplification was introduced in [14]. This system uses eye tracking in combination with VR Headsets. Their main contribution consists of thread creation methods for ray tracing with OpenCL on modern graphics hardware. In contrast to the existing methods our approach concentrates on (a) large projection-based systems and (b) enhancing rendering performance and image quality with global illumination. Our ROI-based rendering technique enables us to render the ROIs at a high quality and the periphery based on a level-of-detail approach, dynamically altering the sampling densities.

3 Framework and Implementation

The basic idea is to provide a framework for user-centered, adaptive high-quality rendering. Therefore, we suggest a number of building blocks taking care of the various tasks that have to be carried out throughout the rendering process. These are described in section 3.1. As a proof of concept and a basic guide, we provide exemplary implementations of each building block and plug them together into a complete system in sections 3.2 to 3.6.

3.1 Building Blocks

Figure 1 shows an overview of the building blocks we suggest. The interaction block is responsible for getting information on how to construct the ROI and also for computing it, which is done in a per-pixel manner. In order to do this, it is necessary to provide a method for determining the actual importance of a pixel. This is done by a user-defined distance measure taking into account the image resolution, the focused pixel coordinates and a set of user-defined parameters like the user's position relative to the display system. Based on this information, the measure then has to determine the distance between each pixel and the ROI. Section 3.2 provides an example for such a distance measure.

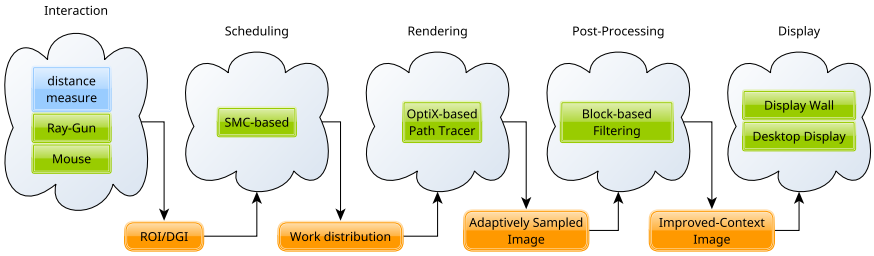


Fig. 1: Building blocks of our suggested framework, exemplary implementations and data flow; the clouds represent the building blocks and contain exemplary implementations represented by green boxes. The orange nodes illustrate the data provided by each of the blocks.

The ROI resulting from the distance computation can be represented with per-pixel values stored as a distance-guide image (DGI); alternatively, it would also be possible to do the actual computation of these values on-the-fly in the scheduling block without an intermediate storage. However, as the ROI is only modified on-demand in our current implementation, it makes sense to store the current DGI instead of recomputing it in each iteration. The DGI is assumed to have the same resolution as the rendered image and stores values between 0 and 1. The scheduling block takes this actual distance information and applies a suitable scheduling algorithm. The scheduling depends on the actual representation of distance values as well as the targeted processor architecture employed for rendering. The generated task distribution is then used to efficiently render an image accounting for the user-defined ROI in the rendering block with a suitable rendering method. As the resulting image is potentially sampled sparsely (see section 3.3), there might be a high variance due to low sample numbers or even unsampled pixels outside the ROI. These areas tend to be perceptually disturbing. In order to increase the visual context's quality for the user, a post-processing block is suggested. Here, image parameters such as brightness can be locally adapted to yield a more homogenous appearance and actual filtering algorithms can be applied. The resulting image is then forwarded to the display-block, the abstract representation of the actual display system, such as a simple desktop display, a projection system or a high-resolution multi-display system with an underlying management middleware. The following subsections provide further information on how the individual components can be implemented by giving a few examples.

3.2 Interaction and Distance Measure

In this section we present the two interaction methods we implemented for selecting a region of interest. Besides the mentioned methods we also experimented with head-tracking, which seemed to be a natural approach to select a region

for convergence. However, this forces the user to stare at the same region until the desired image quality is achieved, which can be anywhere from seconds to hours depending on the scene and the lighting configuration. This is a huge drawback and renders this approach practically useless in the presented context. Obviously, the same argument is also valid for the additional use of eye-tracking.

Ray-Gun-based Tracking Our first interaction method is based on a "Ray-Gun" approach. We use a tracked interaction device (FlyStick) for pointing at the display system and "shooting" a ray through the center of the ROI on demand. The FlyStick's analog stick can also be used to modify the horizontal and vertical fields of view. The distance measure is defined by the projection of an elliptical shape onto the display system, oriented orthogonally to the user's pointing direction. To determine the field of view's (FOV) projection on the display system, the view center and four corner points are used, defining the horizontal and vertical size of the ROI. We determine the pixel coordinates of these points by intersecting rays with a virtual model of the display system. As we make no fixed assumption about the shape of the display system but use an exact geometric model, the projected shape might not be an elliptic or oval shape. In our case, the display wall is curved with a 10° angle between each display column and the resulting DGI looks similar to the one shown in Figure 2. For each pixel in the image, the importance is now determined by computing the minimum distance to any point inside the projected elliptic shape, normalized to $[0, 1]$. All pixels inside the projected shape get assigned the distance value 0, which means that they are sampled in each iteration of the Path Tracing process. A hybrid approach between bisection and Newton's method can be used to find the closest point on the projected shape and thus the shortest distance. For a detailed explanation see [13].

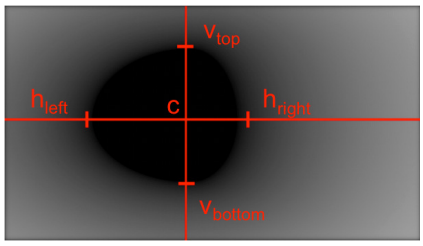


Fig. 2: The Detail-guide Image (DGI) used to guide the rendering process

Mouse-based An ROI can also be defined in a mouse-based manner by simply defining its center with a click. In a projection system, this is more of a fallback method when no tracking is available. However, it can be useful in a

high-resolution desktop environment. Due to the user's lower distance to the display system and the higher pixel density, the aforementioned ROI generation method is also employed in this case. Although, as we assume that there is no tracking system in a desktop environment, the direction for projecting the field of view is perpendicular to the display system's surface, which is considered to be plane. This projection is performed in an orthogonal way, as no distance between the user and the display system is known. Instead, the size of the ROI can be defined in a graphical user-interface when using the mouse.

3.3 Scheduling

Based on the generated DGI, a method has to be developed to compute the sampling densities used during the rendering process in a way so that they are inverse to the DGI's distances. In order to do that, we interpret the distance values $d(x, y) \in [0, 1]$ as inverse probability values for sampling the individual pixels, i.e., $p(x, y) = 1 - d(x, y)$. We then make a binary sampling decision for each pixel individually based on these probabilities and store this decision in a binary image with the same resolution as the rendered image (1 for pixels to be sampled, 0 otherwise). Note due to the probabilistic approach, there is a variation in the number of samples computed between the individual iterations.

This step is necessary because simply computing the samples for pixels with a positive sampling decision would lead to a high thread divergence, thus resulting in idle threads on the GPU. Instead, the binary image computed in the aforementioned step is processed with cuSparse's *Sparse Matrix Compression (SMC)*, which effectively yields the pixel coordinates of non-zero values, i.e., the pixels that have to be sampled. Also, this provides the total number of non-zero values, which means that the Path Tracing process can now be launched with an appropriate number of threads. In turn, the pixel area they have to sample can be directly determined by looking it up in the coordinate array.

Figure 3a shows an unfiltered rendering resulting from SMC-based scheduling. One major issue that arises from rendering an image with these sparsely sampled areas is the potentially high variance in brightness which the human visual system can perceive still well outside the area of sharp vision. Section 3.5 suggests a basic filtering method to overcome this issue, also providing an image brightness similar to the full-resolution image early in the rendering process.

3.4 Rendering

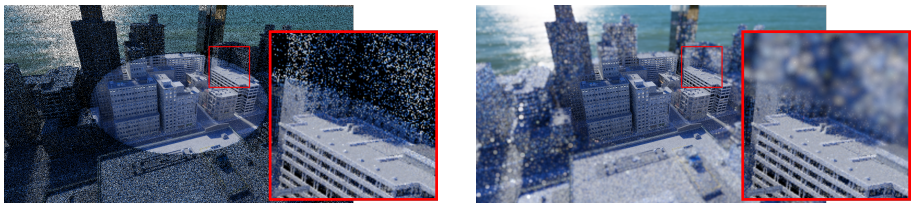
Rendering is performed using our path tracer *Spark*, which is based on NVIDIA's OptiX framework. The rendering process is done completely on the GPU.

3.5 Filtering

As shown in Figure 3a, the varying noise and sampling density outside the ROI can be visually disturbing. We suggest a Gaussian-based filter with varying kernel

size in order to improve the user's visual context. However, the implementation of this filter is still prototypical and does not deliver sufficient performance for a continuous use in an interactive environment. Thus, it serves only as an example for how a filtering approach for an improved visual context could work in general. Note that due to the probabilistic sampling decision made for each pixel, there may always be pixels whose area has not been sampled at all so far. Thus, during the filtering process, only pixels that have been sampled at least once are accounted for, which means that there is no negative influence of unsampled pixels on the image brightness. We use a Gaussian filter kernel with $\sigma = k_i/3$, with k_i being the kernel radius for pixel i. The maximum kernel size k_{max} is set by the user, so that the actual kernel size $k_i = w_i \cdot k_{max} \cdot d_i$ for a pixel i can be computed, based on the distance value d_i contained in the DGI. Also, $w_i = \max\{0, 1 - (s_i/c)\}$ is used to linearly blend between the values from the original image and the filtered results, so that the original, unblurred image becomes more and more visible with the number of samples approaching c. Here, s_i is the current number of samples rendered for the specific pixel and c is a blending constant. This blending constant effectively leads to a slowly decreasing kernel size with the rendering process going on. This also means that image regions are kept at their respective number of samples regardless of whether the ROI is changed.

Instead of performing the blending based on a fixed number of samples, which does not allow for a reliable estimation of the actual noise contained in the image, pixel variance could be employed. Figure 3b shows an example of Gaussian filtering for unconverged, non-focused areas.



(a) An unfiltered image with at most 128 samples per pixel

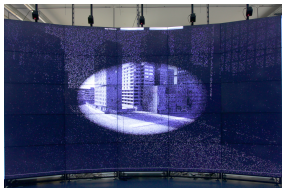
(b) Result of the Gaussian filtering approach applied to the same image

Fig. 3: Unfiltered vs. filtered image: The perceived difference in brightness is clearly visible.

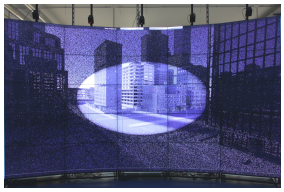
3.6 Display

As mentioned above, we employ our large, high-resolution display consisting of 7×5 1080p displays to mimic a 4k projection system. Figure 4 shows an example of the rendering results displayed on this system with and without filtering. In

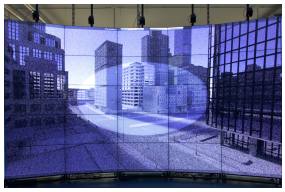
order to bring the 4k rendering to the display wall, the DVI output of a regular PC is connected to a video grabber card, which streams the desktop to the SAGE framework, which is in turn responsible for displaying it.



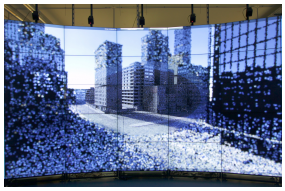
(a) max. 1 sample per pixel, no filtering



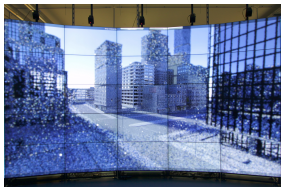
(b) max. 8 samples per pixel, no filtering



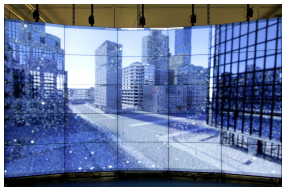
(c) max. 64 samples per pixel, no filtering



(d) max. 1 sample per pixel, Gaussian-based filtering



(e) max. 8 samples per pixel, Gaussian-based filtering



(f) max. 64 samples per pixel, Gaussian-based filtering

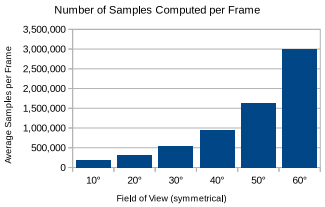
Fig. 4: Comparison between unfiltered and filtered images displayed on our large, high-resolution display system. Filtering leads to a good approximation of the overall image brightness.

4 Benchmarks

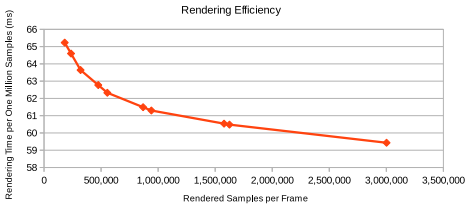
For the presented system, we performed various measurements. First, we are interested in how the decreased ray coherence and density influence rendering efficiency. Thus, we measured the rendering time per one million samples for various fields of view, with larger fields of view corresponding to an increased coherence because of the overall higher sample density. This should in turn lead to a lower thread divergence on the GPU. Also, the pixel density itself is presented. All benchmark results regarding rendering efficiency represent the average of 100 runs.

4.1 Setup

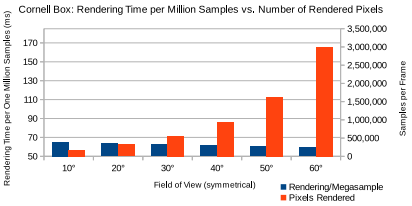
With our current implementation we mainly target 4k projection systems. As we do not have a large-scale 4k projection system available, we mimic it using



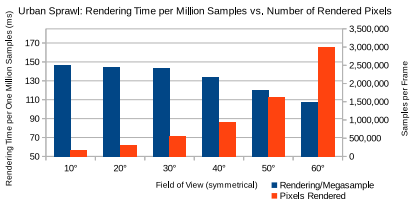
(a) Percentage of samples computed per frame for FOVs from 10 to 60 degrees



(b) Rendering efficiency



(c) Number of computed samples vs. rendering time per one million samples for the Cornell Box scene



(d) Number of computed samples vs. rendering time per one million samples for the Urban Sprawl scene

Fig. 5: Benchmarks: Computed samples for various FOVs, rendering efficiency

our large, tiled display wall by streaming and upscaling a PC's 4k video output to the SAGE framework. For rendering, we use a Xeon E5-2637 with 16GiB RAM, equipped with a GeForce GTX 980 graphics adapter running Linux. As test scenes we use a simple Cornell Box and the more complex city model *Urban Sprawl 2 by Stonemason* (around 350k triangles), both with environmental illumination only. All benchmarks have been performed with the viewpoint set to an exemplary position 1m in front of the display wall, looking towards its center. The rendering resolution is 3840×2160 pixels.

4.2 Results

DGI generation using a separate CUDA kernel at a resolution of 3840×2160 pixels remained almost constant for all fields of view, taking 2-3 milliseconds. Figure 5a shows the relation between the number of samples computed per frame and a varying FOV. Clearly, an increased FOV results in a greater number of samples that need to be rendered per iteration. The influence of this number on rendering performance is shown in Figures 5c and 5d, which show rendering time in milliseconds per one million samples, thus denoting rendering efficiency. It becomes clear that a greater FOV with a consequently higher number of samples usually results in an efficiency increase, as overheads can be reduced and

scheduling efficiency increased, where the latter is strongly related to increased ray coherence and sample density. Nonetheless, even with a reduced rendering efficiency it has to be kept in mind that fewer samples have to be rendered per iteration for smaller fields of view. This means that the focused areas usually still converge much faster than with the original efficiency achieved for the full resolution, as shown below. The general behaviour of rendering efficiency shown in Figure 5b is similar for the Urban Sprawl scene.

We also compared our results to plain Path Tracing without any guiding methods, corresponding to the maximum achieveable rendering efficiency. The pure rendering time for a full 3840×2160 image with one sample per pixel is 460 milliseconds for Cornell Box and 750 milliseconds for Urban Sprawl. Thus, the rendering times per megasample result to 55.46 and 90.42 milliseconds, respectively. For us however, the rendering time per iteration is very important, as this yields some information on how far the image has converged in the focused area after a certain amount of time. Looking at the numbers for several fields of view, we can see that for a small FOV the convergence rate in the focused area is enormous when compared to standard Path Tracing; a 10° FOV with SMC-based scheduling yields a rendering time of 12.29ms vs. 460ms for full-resolution rendering, which is a factor of 37.43, corresponding to approximately 84 % reduced noise according to MC methods' probabilistic error of $\mathcal{O}(1/\sqrt{n})$. For a 60° FOV, SMC-based scheduling yields a rendering time of 188ms vs. 460ms of standard Path Tracing, which is still a factor of 2.44. The numbers for Urban Sprawl are similar.

Note that with the rendering process progressing, the error difference between areas converging at different rates decreases. With a sampling probability of $p_0 = 1$ for pixels inside the ROI and $p_1 \in [0, 1]$ for some pixel outside the ROI, the number of drawn samples after k iterations should be $n_0 = k$ for pixels inside the ROI and $n_1 \approx p_1 k = p_1 n_0$ for the outside pixel. With the MC asymptotic error of $\mathcal{O}(1/\sqrt{n})$ the error difference between pixels inside and outside the ROI is in

$$\mathcal{O}\left(\frac{1}{\sqrt{n_1}} - \frac{1}{\sqrt{n_0}}\right) = \mathcal{O}\left(\frac{1}{\sqrt{p_1 n_0}} - \frac{1}{\sqrt{n_0}}\right) = \mathcal{O}\left(\frac{1 - \sqrt{p_1}}{\sqrt{p_1 n_0}}\right).$$

With p_1 being constant for the specific pixel, this difference approaches 0 with an increasing number of samples.

5 Conclusion and Future Work

We have shown how techniques using ROIs guiding the Path Tracing process can be employed to increase performance and local quality in global illumination rendering with a focus on, but not limited to large, 4k projection and desktop systems. Moreover, we presented an approach to schedule the rendering tasks guided by an ROI on GPUs and presented a prototypical filtering method to combine the computed samples to a final image. To interpret our results correctly, it is important to note that our work is based on the assumption that there

actually is an important area in the image which is desired to be rendered at a high quality, while the quality of the remaining image is not as important. Though, the context of the non-focused area is preserved. We achieved our goal of improving performance for such cases, as shown in the benchmark section, where we also compared our runtime results with standard full-resolution Path Tracing.

Future research should include scheduling methods for increased ray coherence, which may result in a further performance benefit. Sparsely sampled as well as focused areas could be processed with denoising methods such as Ray Histogram Fusion [15], which has recently also been optimized for GPUs [16]. It has to be analyzed whether denoising should be performed at full resolution or at area-specific fractions of the original resolution.

Also, performance and perceptual aspects of such a reconstruction/filtering step have to be taken into account. Even though the basics of gaze-based rendering are well-understood, a user study has to be performed on how the noise introduced by the stochastic nature of Path Tracing affects the perceived quality of rendered images using ROIs based on the user's gaze. Depending on their performance, reconstruction methods could also be implemented to be executed once every few seconds or itereatively for static viewpoints. Generally, the presented work aims for the next step to be the realization of such a system with the full 72 megapixel resolution of our lage display wall. As this rendering should take place on a rendering cluster with several dozens of GPUs, a hybrid scheduling approach for adaptive resolutions is one of the major challenges for achieving this goal.

References

1. Kajiya, J.T.: The rendering equation. In: ACM SIGGRAPH Computer Graphics. Volume 20. (1986) 143–150
2. Robertson, G.G., Mackinlay, J.D.: The document lens. In: Proceedings of the 6th Annual ACM Symposium on User Interface Software and Technology. UIST '93, New York, NY, USA, ACM (1993) 101–108
3. Lamping, J., Rao, R.: Laying out and visualizing large trees using a hyperbolic space. In: Proceedings of the 7th Annual ACM Symposium on User Interface Software and Technology. UIST '94, New York, NY, USA, ACM (1994) 13–14
4. Papadopoulos, C., Kaufman, A.E.: Acuity-driven gigapixel visualization. IEEE Transactions on Visualization and Computer Graphics **19** (2013) 2886–2895
5. E. LaMar, B.H., Joy, K.I.: A magnification lens for interactive volume visualization. In: Computer Graphics and Applications, 2001. Proceedings. Ninth Pacific Conference on. (2001) 223–232
6. Krüger, J., Schneider, J., Westermann, R.: Clearview: An interactive context preserving hotspot visualization technique. Visualization and Computer Graphics, IEEE Transactions on **12** (2006) 941–948
7. Levoy, M., Whitaker, R.: Gaze-directed volume rendering. In: Proceedings of the 1990 Symposium on Interactive 3D Graphics. I3D '90, New York, NY, USA, ACM (1990) 217–223

8. Funkhouser, T.A., Squin, C.H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In: Proceedings of the 20th annual conference on Computer graphics and interactive techniques, ACM (1993) 247–254
9. Yee, H., Pattanaik, S., Greenberg, D.P.: Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. ACM Trans. Graph. **20** (2001) 39–65
10. Guenter, B., Finch, M., Drucker, S., Tan, D., Snyder, J.: Foveated 3d graphics. ACM Transactions on Gra,phics (TOG) **31** (2012) 164
11. Murphy, H., Duchowski, A.T.: Gaze-contingent level of detail rendering. EuroGraphics 2001 (2001)
12. Weaver, K.A.: Design and evaluation of a perceptually adaptive rendering system for immersive virtual reality environments. Master's thesis, Iowa State University (2007)
13. Weier, M., Maiero, J., Roth, T., Hinkenjann, A., Slusallek, P.: Enhancing rendering performance with view-direction-based rendering techniques for large, high resolution multi-display systems. In: 11. Workshop Virtuelle Realität und Augmented Reality der GI-Fachgruppe VR/AR. (2014)
14. Fujita, M., Harada, T.: Foveated real-time ray tracing for virtual reality headset (2014)
15. Delbracio, M., Mus, P., Buades, A., Chauvier, J., Phelps, N., Morel, J.M.: Boosting Monte Carlo Rendering by Ray Histogram Fusion. ACM Trans. Graph. **33** (2014) 8:1–8:15
16. Szeracki, S., Roth, T., Hinkenjann, A., Li, Y.: Boosting histogram-based denoising methods with gpu optimizations. In: accepted for 12. Workshop Virtuelle Realität und Augmented Reality der GI-Fachgruppe VR/AR. (2015)