Analytic Antialiasing for Selective High Fidelity Rendering

Peter Longhurst[*], Kurt Debattista, Richard Gillibrand, Alan Chalmers
University of Bristol,
Department of Computer Science,
Merchant Venturers Building,
Woodland Road,
Bristol. BS8 1UB, UK

Abstract

Images rendered using global illumination algorithms are considered amongst the most realistic in 3D computer graphics. However, this high fidelity comes at a significant computational expense. A major part of this cost arises from the sampling required to eliminate aliasing errors. These errors occur due to the discrete sampling of continuous geometry space inherent to these techniques. In this paper we present a fast analytic method for predicting in advance where antialiasing needs to be computed. This prediction is based on a rapid visualisation of the scene using a GPU, which is used to drive a selective renderer. We are able to significantly reduce the overall number of anitialiasing rays traced, producing an image that is perceptually indistinguishable from the high quality image at a much reduced computational cost.

1 Introduction

High-fidelity rendering is responsible for producing the highest quality perceivable images simulated through physically-based illumination of a virtual scene. Rendering such images is notoriously computationally expensive through having to fully solve the rendering equation [15]. Ray-tracing [36] and its extensions, particularly variants of stochastic ray-tracing [6, 15], have long been the algorithms of choice for solving the rendering problem. For each pixel in the screen a number of samples are shot and the radiance is calculated for each one resulting in lengthly execution times. The computation cost is further accentuated since certain areas of the resulting images require more samples than others per pixel in order to reduce aliasing artifacts that arise from areas of high spatial frequencies. Quality deficiencies (aliasing errors) are highly noticeable if a particular pixel contribution is not sufficiently sampled. These errors occur if the data collected for a pixel is incomplete due to insufficient sampling.

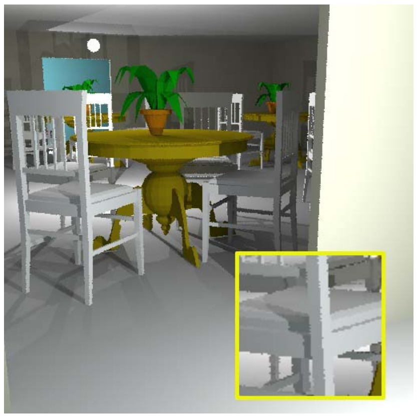

Figure 1. Under sampled global illumination image demonstrating aliasing artifacts.

Figure 1 shows a rendered scene containing aliasing errors. The majority of these, known as *jaggies*, occur at edges in the scene. Particularly obvious are the errors on the back of the chair in the foreground. This image was created using the lighting simulation system *Radiance* [34]; only one jittered ray was traced for each pixel. Unfortunately, tracing more sample rays for each pixel to improve the quality of the result increases computation linearly. Several common sampling methods exist that are used widely in global illumination applications. These include supersampling, adaptive sampling [36], and stochastic sampling [5]. All of these methods, however, are over conservative as they operate with no knowledge of the actual content of the scene; pixels where no aliasing would occur are still sampled to a certain extent.

When considering ray tracing, the number of samples

[*]e-mail: pete.longhurst@bristol.ac.uk

needed per pixel can be adjusted based on the complexity of that area of the image. Complex areas where many objects contribute to the image may require many samples to deal with antialiasing. However, a much lower number of rays may be traced for areas occupied by a single surface. An alternative to actually rendering the scene, is to develop a method to extract the necessary pixel level information from the scene geometry. This technique should be fast, at least relative to rendering the final result, otherwise the advantages of selective rendering are nullified. An estimate of the resultant image would indicate the sampling requirement for each pixel and unnecessary rays would not need to be traced thus significantly reducing rendering times.

Fortunately, modern graphics hardware provides, albeit at a reduced quality, the ability to rapidly generate locally illuminated images of a scene. This paper investigates how an aliasing error prediction map, which we term *aamap* can be used to selectively render a high-fidelity image only supersampling pixels when necessary. The *aamap* is achieved by using a GPU to first rapidly estimate the image using a traditional rasterisation pipeline, a technique we term *Snapshot*. Subsequently we generate the *aamap* on the GPU itself gaining a speedup over the CPU had it been used for this image processing calculation. This information is then used to correctly direct our selective renderer, an extension of the lighting simulation package *Radiance*, to produce perceptually high quality images at significantly reduced computational times. We demonstrate the speedup for selective rendering two scenes under a number of anitialiasing settings and furthermore demonstrate, by the use of a Visible Differences Predictor (VDP) [8], that there is no perceptual difference between the supersampled version and our novel technique.

This paper is divided as follows. Section 2 presents related work in the field. Section 3 introduces the quick image estimate and discusses the generation of the *aamap* using the GPU. Section 4 presents our adaptive antialiasing selective renderer. Section 5 presents speedup for two scenes. Section 6 compares the supersampled rendering with our approach using the VDP. Finally, Section 7 presents conclusions and future work.

2 Related Work

In this section we discuss related work in antialiasing, the GPU as a method of accelerating high-fidelity rendering and finally, since we use VDP to evaluate our results we present related work in that area.

2.1 Antialiasing

Antialiasing was first identified by Crow to be a major source of artifacts in computer generated images [7]. Since then, there has been significant work on antialiasing for ray tracing applications, for example see [5, 9, 24, 27, 31].

Previous work to accelerate the processes of antialiasing and primary ray calculation has included, using visible surface preprocessing to create view and/or light buffers to determine which areas of the image contain objects or are in shadow since these methods are generally much faster than the rendering process [3, 12] and lend themselves to implementation in hardware [38]. Antialiasing is achieved by subdividing pixels and hence effectively storing a higher resolution image. The aliasing effects still appear however for objects and edges that are smaller than the higher resolution level.

Salesin and Stolfi [30] proposed the ZZ Buffer which stores tile lists for cells of pixels. Each tile list contains a linked list of objects visible for that cell and the maximum and minimum depth of the list. In addition, each tile in the list contains the data from a single object clipped to fit the cell and its maximum and minimum depth. The ZZ Buffer produced is then scanned at render time to determine the complexity of the tiles; simple tiles are sampled with a single ray whereas complex tiles are super-sampled with a uniform number of rays to produce anti-aliasing. Again this super-sampling approach leaves the possibility of missing sub-pixel objects that occur between the sampling rays.

Adaptive super-sampling [18] used statistical analysis to determine, for each pixel, when enough rays have been traced by determining when the contribution of additional rays is insignificant. While this is an effective method of producing high quality images there is the problem of not knowing how many rays are going to be needed and hence complex scenes can have a significant rendering cost.

2.2 Programmable Graphics Hardware

The performance of modern GPUs has risen to tens of Gflops well above the performance of CPUs for certain types of application [2]. While GPUs can be used for a numerous number of rasterisation based techniques [10] our primary interest is in their use as an aid to high-fidelity rendering. GPUs have been used to accelerate ray-tracing renderers in a number of distinct methods including hybrid CPU and GPU ray-tracers [4], GPU only ray-tracers [28] and for general purpose computation [2] which can be tied into ray tracing. These systems are however not fully-fledged physically-based renderers. GPUs have been used for quick image estimates for generating saliency maps that can be used by the high-fidelity renderer [20, 39]. The estimates are then used as input to a saliency map [13, 14, 16] generator. In these systems the GPU does not need to do any extra processing apart from the standard rasterisation pipeline. Our approach differs from all the above in that it not only assists a high-fidelity renderer but also we exploit the fast image processing availabilities of the GPU itself to generate the *aamap*

2.3 Perceptual Image Difference Techniques

Image quality metrics have been developed using psychophysical approaches and computational techniques to differentiate between pairs of images [17, 22, 26, 29, 35]. There are numerous metrics that can be applied to a compare a pair of images, however if a perceptual visual difference is required, the metric must somehow consider the workings of the human visual system (HVS). The two main approaches that fulfill this criterion are the Sarnoff Visual Discrimination Model (VDM) [1, 21] and Daly's Visual Difference Predictor (VDP) [8, 25]. There have been several evaluations of these measures [19, 23, 40].

3 The Snapshot

Snapshot produces a rapid image estimate from the scene description. We chose OpenGL as a basis for our analytical antialiasing technique because it is well supported in hardware, it is fast and portable. *Snapshot* is designed to read model data from a Wavefront ".obj" file. This is an established format which many renderers can read. Lighting information is read in from a simple custom light file format which states position and emission properties. Camera positions are handled through the same view files used by *Radiance*. Multiple views can be stipulated in one of these files to represent an animation.

Shadows and reflections are a key component of any high fidelity image and thus it is important that these too are antialiased correctly. To predict the shadow boundaries, we generated shadows in our scenes by projecting shadow maps from the front and back of each light source [37]. Mirror reflections are generated by redrawing the scene's geometry from a reflected camera position onto the mirror plane. Figure 2 illustrates correct shadows and reflections generated from our system. It would be possible to predict more general reflections and also refractions within *Snapshot* using a technique such as cubic environment mapping [33]. This is something we will consider in the future; it is however debatable how much errors in these regions would detract from a final image.

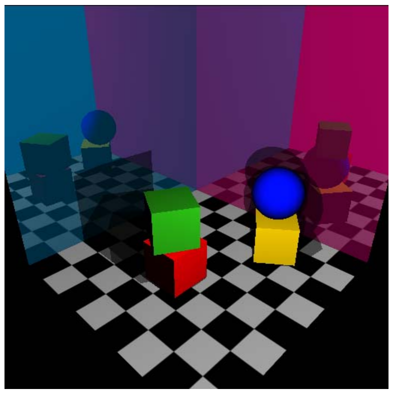

Figure 2. Snapshot with texture mapping, shadowing and reflections.

Common OpenGL hardware extensions were used to offload the generation of the shadows, and to perform high resolution off-screen renders. A potential cost of creating the *aamap* was the time required to read the result back from the screen. We were, however, able to keep this cost to a minimum by using a graphics card built for PCI-Express bus architecture. This is a new standard which allows high speed bi-directional communication with PC cards.

Initially the *Snapshot* is created based on all the information available for the scene. Pixel prioritisation can then occur using the *aamap* generated from this preview. Additional information such as task relevant pixels can also be introduced at this stage [32].

3.1 Creation of the Pixel Prioritisation Map

Aliasing artifacts appear in a rendered image typically at the edges of objects and shadows. The strategy for identifying where these errors occur makes use of a high pass filter, or edge detector. An initial consideration may be to use a filter in fourier space, however the key feature of the generation of our map is the speed that can be gained from good use of the GPU. Although an FFT can be performed relatively quickly on recent CPUs, for this application, image space filtering is preferable as image convolution filters lend themselves well to implementation on graphics hardware. We used a standard 5×5 Laplacian filter as we found this filter good at picking up both low contrast edges and high frequency texture information. Other filters which are instead gradient based, such as the Sobel or Prewitt, would work equally well. Future work will consider a combination of components from a Haar wavelet decomposition as an alternative to the Laplacian. Figure 4 compares the time required to filter an image with a 5×5 filter kernel on a standard CPU and on a modern graphics card. It is clear from

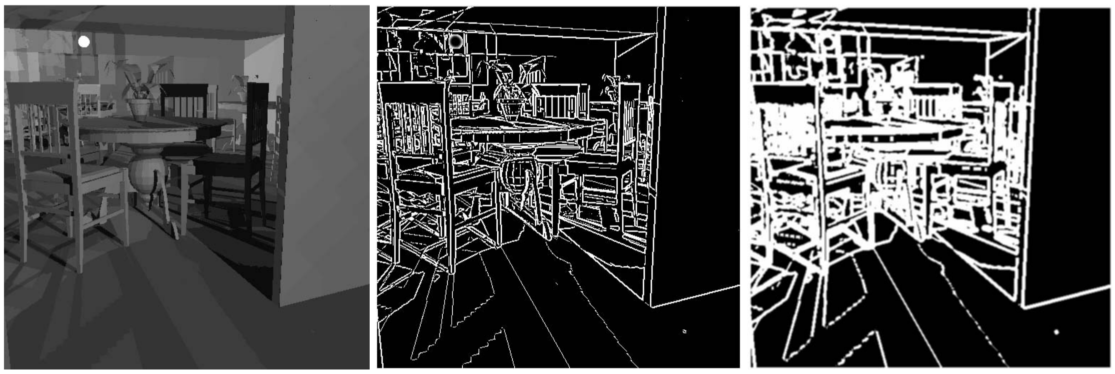

Figure 3. Greyscale Snapshot (left), Laplacian edge map (middle), and *aamap* (right).

the graph how even a current, high performance, processor cannot come close to matching the raw pixel processing power of the graphics card. The advantage that a GPU has over a conventional CPU for this type of application is the inherent parallelism. The 6600GT that we use, for example, has 12 pipelines allowing for 12 pixels to be filtered concurrently. Due to this, in this comparison the GPU is consistently at least 5 times faster regardless of resolution.

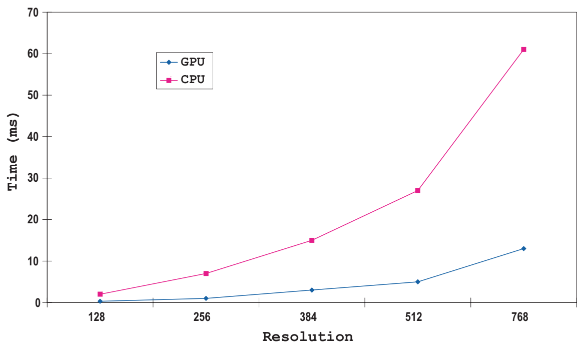

Figure 4. Linear image filtering: Nvidia 6600GT GPU vs. P4 3.4Ghz CPU.

To undertake the required filter on the graphics hardware we used a fragment shader written in *CG*. *CG* is a language developed by Nvidia, that works in conjunction with either DirectX or OpenGL, to program certain aspects of the graphics pipeline [11]. A fragment shader can be simply thought of as a small program that operates once per fragment (in this case a fragment equates to a pixel). To use a shader to computer image filtering we first draw the scene as normal and record this into a texture. The next step is to create a square which exactly fits the screen. Our shader modifies each pixel of this square as it is drawn with values from the texture image. The resulting image (the filtered result) can then be read into a second texture, which, in turn can be passed as an image into our selective renderer. In this way it is possible to recreate almost any local image filter on a modern GPU.

Figure 3 shows the intensity image from *Snapshot* along with the initial edge map and the final *aamap*. Because the edges in the initial Laplacian edge map themselves suffer from aliasing, we blurred the edges using an averaging filter to construct the *aamap*. Additionally, given the relatively low cost of the generation of these maps (see figure 4) it was possible for us to increase their accuracy by creating them at a larger resolution than the final render.

4 Selective Rendering

In order to demonstrate the effectiveness of our approach we have extended the lighting simulation system *Radiance* renderer `rpict` to a new renderer we term `apict`. `apict` performs jittered stratified sampling. A user-defined variable sets the level of stratification per pixel effectively dictating the number of rays shot per pixel. `apict` can be passed as a parameter the *aamap* in the form of a grey-level map generated by *Snapshot*. The *aamap* dictates to `apict` two methods for sub-sampling. Firstly, the grey-level in the map modulates the stratification parameter, from one to the stratification value defined by the user. Secondly, the map is used to direct adaptive subsampling. The adaptive subsampling shoots a ray for every fixed number of pixels in both the x and y direction. The distance between rays shot is given by a user parameter. A common value is 8 or 16. Whenever the four rays outlining an area are shot, the value of the pixels in *aamap* corresponding to that same area are consulted. If the total of their values is greater than a user-specified value (typically 0, meaning the map must be completely black in this area), the area is subdivided into four and the same procedure is applied recursively until the area corresponds to one pixel. When the total value of the area in the *aamap* is less than or equal to the threshold, the values

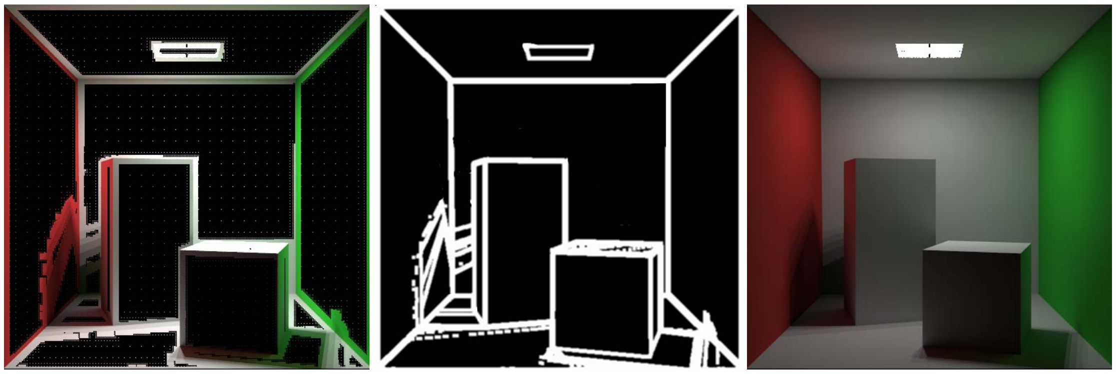

Figure 5. Interpolated adaptive subsampling (left), *aamap* (middle), and result (right)

within this area are interpolated. Figure 5 (left) illustrates which rays are shot for the adaptive subsampling algorithm given an *aamap* (middle) and the final image (right) shows the final interpolated solution.

5 Results

We show timings for two scenes to demonstrate the effectiveness of our approach. All timings shown in this section were achieved on a dual Intel Xeon 2.4GHz with 3GB of ram under Linux. The *aamap* timings are based on an Nvidia 6600GT AGP graphics card.

We used a Cornell box scene and a scene designed to contain a large number of edges (Figure 3). To simulate the area light source used within the Cornell box we subsampled the light source with five point light samples. This was necessary as it is not currently possible to create area light sources in OpenGL environments. This scene took less than 30ms to create a *Snapshot* of a 768×768 frame on the GPU.

The second scene contained ≈130,000 triangles and 6 light sources, this required 13 passes within *Snapshot* (a first pass and two passes per light source) totaling 155ms for a 768×768 frame. We used a brute force approach to create our images; this time could easily be improved however through some simple OpenGL optimisations, for example invisible object culling. All final timings for both scenes relate to a rendering a 512×512 final frame.

The timing results for our Cornell box test are shown in table 1. The time taken to create the *aamap* is included however, given the simplicity of the scene, this had a negligible impact on the overall time. We compared times for the baseline cases where the same number of rays were traced for every pixel (rpp), against selectively rendering using our map. The *aamap* defined the number of rays traced for each pixel from 1 ray up to the fixed maximum. Additionally we

rpp	**Time (s)**			
		SELECTIVE		
	STANDARD	*Non-Adaptive*	*Space 8*	*Space 16*
1	6.53		1.74	1.75
4	24.52	7.33	2.46	2.43
16	96.46	21.07	10.23	10.15
36	216.21	40.71	22.18	22.18

	Speedup		
rpp	*Non-Adaptive*	*Space 8*	*Space 16*
1		3.8	3.7
4	3.3	10.0	10.1
16	4.6	9.4	9.5
36	5.3	9.7	9.7

Table 1. Timings for the Cornell box (rpp=rays per pixel)

used the *aamap* with adaptive sampling to interpolate at 8 and 16 pixel spacing. For this scene we were able to benefit greatly from using the map. For the selective sampling case where the stratification level is 36 rays per pixel the resultants image is indistinguishable from the reference high quality image. However the selectively rendered image was rendered more than 5 times as fast. Using the adaptive sampling method in conjunction with selective rendering gave us a further performance boost. Combining these strategies we were able to reduce the rendering time by up to 1 order of magnitude. This allowed us, in the time taken to perform a standard render using 4 jittered rays per pixel, to selectively render an image comparable to one with 9 times as many rays traced (36rpp).

Table 2 shows the timings we achieved for our second scene. Again the timings for images which were selectively rendered include also the time taken to generate the *aamap*. Given the complexity of the scene we did not witness such a large reduction in rendering time, however we were able to get a speed increase of almost 3 times. For both scenes we witnessed a linear relationship between the render time and the number of rays traced per pixel (Figures 6 and 7).

rpp	Time (s)			
	STANDARD	SELECTIVE		
		Non-Adaptive	*Space 8*	*Space 16*
1	92.12		53.44	52.20
4	358.06	168.91	130.61	130.80
16	1464.79	586.98	533.35	549.13
36	3217.32	1316.09	1092.37	1101.55

	Speedup		
rpp	*Non-Adaptive*	*Space 8*	*Space 16*
1		1.7	1.8
4	2.1	2.7	2.7
16	2.5	2.7	2.7
36	2.4	2.9	2.9

Table 2. Timings for the room scene (rpp=rays per pixel)

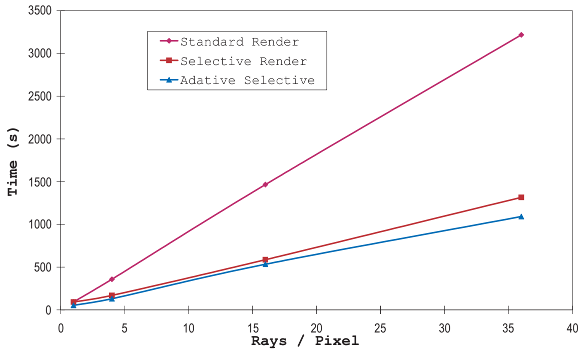

Figure 6. Timing results from the Cornell box

These graphs also show that the relative performance gain increases with the number of rays traced per pixel.

6 Perceptual Assessment of Results

For each scene the full solution was compared to each selective render. These image pairs were evaluated using two different metrics. To find pixel level errors, the mean square error was found as a difference map then averaged for all pixels. Secondly we used an implementation of Daly's visual difference predictor to find the perceivable difference between the images [8]. Primarily each image is treated individually to remove frequencies that would not be witnessed by a human observer. The remaining differences are then weighted with regards to a multitude of frequency and orientation channels. The metric is designed to highlight, as a probability map, differences near and below a just noticeable threshold. Figure 8 shows the room scene rendered both using the full solution and selectively at the highest quality we tested (36rpp) and a resultant VDP difference image. These predicted differences are, for display purposes overlayed in red on the standard image. To gain a single error value for each image we average the map. Although this averaging gives a number which is comparable between the different cases, unfortunately it does not relate directly to the probability of detection of errors (the same average could relate to an imperceivable error throughout the image or a highly perceivable error in one location). For every image this error was less than 1%. A more useful statistic is the percentage of pixels which are in error, as shown in table 3.

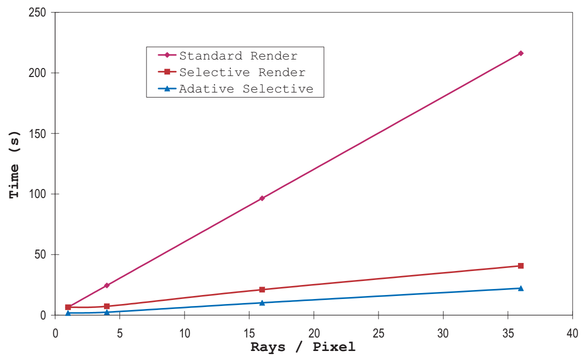

Figure 7. Timing Results from the room scene

rpp	% Error Pixels: Cornell Box					
	NON-ADAPTIVE		SPACE 8		SPACE 16	
	VDP	*MSe*	*VDP*	*MSe*	*VDP*	*MSe*
1			2.94	1.16	4.15	1.98
4	2.74	1.61	3.39	1.91	4.77	2.78
16	1.16	0.54	1.80	0.43	3.53	1.76
36	1.08	0.35	1.17	0.71	3.42	1.49

rpp	% Error Pixels: Room Scene					
	NON-ADAPTIVE		SPACE 8		SPACE 16	
	VDP	*MSe*	*VDP*	*MSe*	*VDP*	*MSe*
1			2.55	8.62	2.57	10.97
4	1.21	9.91	1.32	10.83	1.29	11.19
16	1.00	11.47	1.21	12.25	0.39	12.05
36	0.28	10.60	0.29	37.96	0.30	38.61

Table 3. Selective quality percentage error

For every image we selectively rendered we observed a correlation to the standard render with less the 5% of the image's pixels in error. The Cornell box scene measured slightly worse, this is due to the area light source sampling. Future work will investigate various alternative methods for simulating area light sources from *Snapshot*. However for the 36 rays per pixel case we were able to render scenes perceptually identically to the standard case; neither scene had a significant error at this level of detail.

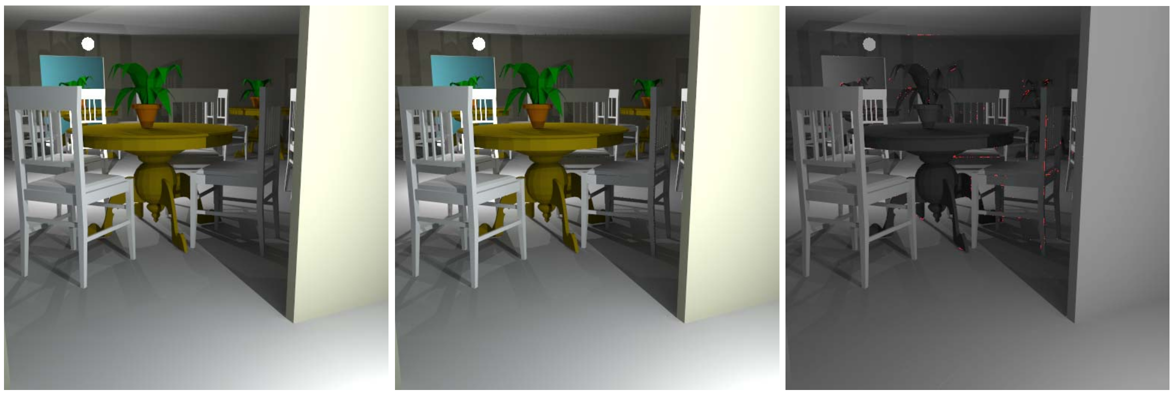

Figure 8. High quality render (left) vs Selective render (middle), VDP difference overlayed in red (right)

7 Conclusions

We have successfully shown that selectively rendering a scene based on our *aamap* can significantly reduce the time taken to render the scene. Furthermore, we have shown the resultant image to be perceptually indistinguishable from the same image rendered traditionally. For our simple scene we have achieved a speed up greater than one order of magnitude. For our more complex scene, although we may not achieve such an improvement, we still achieve a good speedup. Using *Snapshot* we have demonstrated the ability to correctly predict areas where sampling is required to reduce aliasing artifacts. Our selective renderer `apict` is able to use this information to combine two separate sampling strategies to reduce the time take to render the scene.

Future work will extend the process outlined in this paper to include temporal antialiasing for animated sequences. Furthermore we intend to extend the work from [20, 39] to be more inline with our philosophy of hybrid GPU-CPU rendering, whereby the relatively expensive saliency map computations [14] will be computed on the GPU.

References

[1] M. R. Bolin and G. W. Meyer. A visual difference metric for realistic image synthesis, 1999.

[2] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. Brook for gpus: stream computing on graphics hardware. *ACM Trans. Graph.*, 23(3):777–786, 2004.

[3] L. Carpenter. The a -buffer, an antialiased hidden surface method. In *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 103–108. ACM Press, 1984.

[4] N. A. Carr, J. D. Hall, and J. C. Hart. The ray engine. In *HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 37–46, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[5] R. L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):51–72, 1986.

[6] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 137–145, New York, NY, USA, 1984. ACM Press.

[7] F. C. Crow. The aliasing problem in computer-generated shaded images. *Commun. ACM*, 20(11):799–805, 1977.

[8] S. Daly. The visible differences predictor: An algorithm for the assessment of image fidelity, 1993.

[9] M. A. Z. Dippe and E. H. Wold. Antialiasing through stochastic sampling. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 69–78. ACM Press, 1985.

[10] R. Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[11] R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[12] E. A. Haines and D. P. Greenberg. The light buffer: a shadow testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, 1986.

[13] L. Itti and C. Koch. A saliency-based search mechanism for overt and covert shifts of visual attention, 2000.

[14] L. Itti, C. Koch, and E. Niebur. A model of saliency-based visual attention for rapid scene analysis. In *Pattern Analysis and Machine Intelligence*, volume 20, pages 1254–1259, 1998.

[15] J. T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM Press.

[16] C. Koch and S. Ullman. Shifts in selective visual attention: towards the underlying neural circuitry. In *Human Neurobiology*, volume 4, pages 219–227, 1985.

[17] C. Lambrecht and J. Farrell. Perceptual quality metric for digitally coded color images, 1996.

[18] M. E. Lee, R. A. Redner, and S. P. Uselton. Statistically optimized sampling for distributed ray tracing. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 61–68. ACM Press, 1985.

[19] P. Longhurst and A. Chalmers. User validation of image quality assessment algorithms. In *Theory and Practice of Computer Graphics 2004 Proceedings*, pages 196–202, 2004.

[20] P. Longhurst, K. Debattista, and A. Chalmers. Snapshot: A rapid technique for driving a selective global illumination renderer. In *WSCG 2005 SHORT papers proceedings*, pages 81–84, 2005.

[21] J. Lubin. A visual discrimination model for imaging system design and evaluation. *Vision Models for target detection and recognition*, pages 245–283, 1995.

[22] A. McNamara, A. G. Chalmers, T. Troscianko, and I. Gilchrist. Comparing real and synthetic scenes using human judgments of lightness. *Proceedings 11th Eurographics Workshop on Rendering*, 2000.

[23] G. Meyer, H. Rushmeier, M. Cohen, D. Greenberg, and K. Torrence. An experimental evaluation of computer graphics imagery. *Transactions of Graphics*, 5(1):30–50, 1986.

[24] D. P. Mitchell. Consequences of stratified sampling in graphics. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 277–280. ACM Press, 1996.

[25] K. Myszkowski. The visible difference predictor: Applications to global illumination problems. *Proceedings of The Eurographics Workshop on Rendering*, pages 223–236, 1998.

[26] W. Osberger, A. Maeder, and N. Bergmann. A technique for image quality assessment based on a human visual system model, 1998.

[27] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 281–288. ACM Press, 1989.

[28] T. J. Purcell, I. Buck, W. R. Mark, and P. Hanrahan. Ray tracing on programmable graphics hardware. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 703–712, New York, NY, USA, 2002. ACM Press.

[29] M. Ramasubramanian, S. N. Pattanaik, and D. P. Greenberg. A perceptually based physical error metric for realistic image synthesis, 1999.

[30] D. Salesin and J. Stolfi. The ZZ-buffer: A simple and efficient rendering algorithm with reliable antialiasing. In *Proceedings of the PIXIM '89 Conference*, pages 451–66, Hermes Editions, Paris, France, 1989.

[31] A. Schilling. A new simple and efficient antialiasing with subpixel masks. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 133–141. ACM Press, 1991.

[32] V. Sundstedt, A. Chalmers, K. Cater, and K. Debattista. Top-down visual attention for efficient rendering of task related scenes. In *Vision, Modeling and Visualization*, 2004.

[33] D. Voorhies and J. Foran. Reflection vector shading hardware. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 163–166, New York, NY, USA, 1994. ACM Press.

[34] G. J. Ward. The radiance lighting simulation and rendering system. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 459–472. ACM Press, 1994.

[35] B. Watson, A. Friedman, and A. McGaffey. Measuring and predicting visual fidelity. In *Siggraph 2001, Computer Graphics Proceedings*, 2001.

[36] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[37] L. Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.

[38] S. Winner, M. Kelley, B. Pease, B. Rivard, and A. Yen. Hardware accelerated rendering of antialiasing using a modified a-buffer algorithm. *Computer Graphics*, 31(Annual Conference Series):307–316, 1997.

[39] H. Yee, S. Pattanaik, and D. P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. In *ACM Transactions on Graphics*, pages 39–65. ACM Press, 2001.

[40] H. Zhou, M. Chen, and M. F. Webster. Comparative evaluation of visualization and experimental results using image comparison metrics. *Proceedings of the conference on Visualization '02*, pages 315–322, 2002.