An Analysis of Cache Awareness for Interactive Selective Rendering

Piotr Dubla
Warwick Digital Laboratory,
University of Warwick, England
P.B.Dubla@warwick.ac.uk

Alan Chalmers
Warwick Digital Laboratory,
University of Warwick, England
A.G.Chalmers@warwick.ac.uk

Kurt Debattista
Warwick Digital Laboratory,
University of Warwick, England
K.Debattista@warwick.ac.uk

Abstract

Interactive high-fidelity rendering is one of the major goals of computer graphics. Algorithms based on ray tracing are usually used to drive high-fidelity renderers. While ray tracing is often thought to be impractical for real time performance, recent algorithmic advances in the field have led to the development of interactive ray tracers which leverage the performance of modern parallel systems and cache awareness to obtain real-time rates for moderately complex scenes using Whitted-style ray tracing. Another method used for accelerating ray tracing is the use of selective rendering where only the pixels that need to be computed are traced and the remainder are computed by other means such as interpolation. The choice of which pixels to render may depend on a number of factors, from simple ones that just compare the radiance values of the rays within a certain area and shoot further rays recursively if the difference in radiance is above a certain threshold, to more complex ones based on visual attention. Selective rendering algorithms may not be perfectly compatible with current interactive ray tracing techniques since selective rendering methods tend to be naturally incoherent, and the computation of rays at a stride may result in expensive cache misses. In this paper we analyse the effect of selective rendering on interactive ray tracing and hint at possible solutions that would allow selective rendering to be compatible with interactive rendering methods to further improve rendering speeds.

Keywords: Interactive Ray Tracing, Selective Rendering

1 INTRODUCTION

In recent years the demand for interactive high-fidelity graphics has grown tremendously in the fields of rendering and visualisation. This increase in demand coupled with increase in multi-core architectures for CPUs as well as the generalisation of GPUs has once again revitalised ray tracing and brought it to the forefront of high-fidelity rendering and complex visualisation. With today's top-end monitors running natively in HD resolutions such as 1920×1080 it is more important than ever to utilise the increased processing power wisely and direct the usage of computational power on components of the image that require it most.

Current state-of-the-art interactive ray tracing (IRT) implementations are now running on consumer desktop PCs and attention has now shifted to more complex effects such as advanced materials, highly complex scenes and global illumination. Recent advances make maximal use of spatial and cache coherency [RSH05, SSK07, Wal07] along with optimised acceleration structures [WH06, LYTM06, WBS07, WK06]. It is due to the clever exploitation of spatial and cache coherency that current ray tracers have finally been able to achieve interactive frame rates on commodity hardware.

Selective rendering (SR) which encompasses progressive [CCWG88], adaptive [Deb06], time-constrained and perceptually-driven techniques [CCW03, SCCD04] is an approach that concentrates work on pixels in the image that most benefit from the added computational power. This is done by not computing all the pixels in an image, but only those that are deemed necessary. In the case of perceptually-driven methods the pixels in the image that are classified as salient and perceptually important to an observer are where computational resources are focused. The rest of the pixels that are not rendered are generally interpolated in some way. This means that, unlike the cache-coherent IRT methods, selective rendering is naturally incoherent as the rendered pixels are typically distributed all across the image.

While a large body of work exists on selective rendering very little research has been performed on how it functions in an interactive setting and if computing less pixels translates into less overall computational time. The goal of this paper is to simulate different levels of selective rendering in a state-of-the-art interactive ray tracing framework and see how current techniques and selective rendering function together. We will try to determine if the speed-up offered by selective rendering is not offset by the decrease in cache and spatial co-

herency that naturally occurs when only specific pixels in the image are being rendered. This is of great importance as current packet-based ray tracing techniques are heavily reliant on very coherent packets of rays to achieve the levels of performance that offer interactivity on consumer desktop PCs. We conduct a series of experiments which identify future courses of action.

This paper is divided into the following sections: Section 2 presents related work in the fields of selective rendering and interactive ray tracing. Section 3 details our experimental framework used to achieve our results and Section 4 and 5 provide both the results as well as an in depth discussion of their implications.

2 RELATED WORK

In this section we will discuss related work on interactive ray tracing as well as selective rendering.

2.1 Selective Rendering

Selective rendering is defined by Debattista [Deb06] as "those techniques which require a number of rendering quality decisions to be taken and acted upon prior to, or even dynamically during the actual computation of any image or frame of an animation". This covers a large number of approaches and algorithms including adaptive, progressive ones but we will be focusing mainly on perceptually assisted rendering and it use to accelerating ray tracing. Other areas that deal with selective rendering such as refining level of detail (LOD) were initially done by [Cla76]. A thorough summary of more recent real-time techniques is provided in [LWC+02]. Progressive radiosity as implemented by [CCWG88] can also be viewed as a selective rendering algorithm along with more advanced implementation such as the adaptive progressive refinement ray tracer by Painter and Sloan [PS89]

With regard to perceptually assisted selective rendering, approaches such as adaptive sampling, that apply perceptual considerations, were examined by Mitchell as early as 1987 [Mit87]. Perceptually-driven radiosity and ray tracing have been implemented in a number of different ways. A good overview of early of radiosity-based methods can be found in [Pri01]. [BM98] and [FSPG97] produced frequency-based ray tracers that used very complete models of the human visual system and incorporated aspects such as spatial processing and visual masking. Visual difference predictors have also been used to direct samples in stochastic ray tracing as well as determine stopping conditions [Mys98, BM98]. These visual difference predictors were costly to compute though as they had to be re-calculated many times each frame until [RPG99] decoupled their spatially-dependant saliency component from the luminance dependant component. This led to many selective rendering implementations that used saliency models such as [YPG01] where a saliency model they term the *Aleph Map* was used to influence the search radius for samples when performing the indirect-diffuse lighting calculations from an irradiance cache. [HMYS01] utilised saliency maps and task objects to identify the most salient objects on which the glossy and specular components where rendered in higher detail. In [CCW03, SCCD04] saliency maps and task maps where used to vary the number of samples calculated in a global illumination framework based on the *Radiance* renderer by Ward [War94]. Sparse sampling methods such as [WDP99] and its refinement in [WDG02] also use adaptive techniques to focus computation in areas of importance and re-use computations to to provide approximations of lighting in less important areas. Newer adaptive algorithms such as [WFA+05] and its extension [WABG06] provide a framework for rendering a large number of complex lighting effects by adaptively gathering or refining clusters of lights and approximating their overall contributions cheaply.

2.2 Interactive Ray Tracing

While ray tracing has been a favoured research topic since the early 80's [Whi80, CPC84] only recently have methods been presented that have allowed ray tracing to to become interactive on desktop PCs. The first real-time implementations were by Muus [Muu95] and Parker et al. [PMS+99]. They parallelised the traditional ray tracing pipeline to run on shared memory machines, but the computer used at the time were classified as super computers running as many as 96 processors. While both implementation used ray tracing they had different goals. For Muus these were to render CSG directly while Parker et al. implemented a more conventional system that was designed to render complex scenes with millions of triangles. An increase in computational power was needed though before standard desktop systems could support interactive frame rates. This is due to the fact that ray tracing is computationally expensive and as yet has not benefited from commercial purpose-built hardware, although attempts are being made [SWS05]. This increase in computational desktop power only occurred recently when multi-core CPUs entered the market. Initial implementations after Muus and Parket et al. were limited to simple shading effects [RSH05] or to very low resolution images that were rendered not on a single PC but using a distributed environment [WSBW01].

Recent work has attempted to make use of the increase in computational power via two avenues: exploitation of cache coherency and optimisation of acceleration structures.

Previously ray/object intersections were a bottleneck in the ray tracing pipeline. In addition, the actual traversal of the acceleration data structures was also one of the major challenges. With the resurgence of shared-

memory machines containing multiple processors and the progress made with SIMD instruction sets such as Intel's SSE, the exploitation of spatial and cache coherency was paramount in allowing ray tracers to reach interactive rates. The primary method that exploited both spatial and cache coherency was the grouping of rays that have a common origin in bundles or packets, first done by [WSBW01]. These packets also helped in optimising acceleration structure traversal and allowed for early culling of large number of rays. Furthermore Wald et al. [WSBW01] demonstrated that organising data structures carefully to improve the chances they would remain in CPU cache along with the packetisation in ray tracing was the final push that was needed to allow real-time ray tracing on desktop PCs. Due to the constantly increasing bandwidth and decreasing latency to on-die CPU cache (L1, L2 and L3 cache) retrieving data from it was orders of magnitude faster than doing so from main memory.

While exploitation of cache coherency is vital, the research that has gone into construction of efficient data structures that minimise the cost of ray/object intersections cannot be overlooked. These structures, commonly known as acceleration data structures, generally subdivide space in some uniform or adaptive manner and attempt to make traversal as cache coherent as possible. Much research has gone into these structures and while it is impractical to cover them all here the current favourites are kd-trees [WH06], BVHs [LYTM06, WBS07] and a new hybrid approach known as bounding interval hierarchies (BIH) [WK06].

Recent publications such as the one by Wald et al. [WMG+07] provides an in-depth overview of this growing field of research.

3 EXPERIMENTAL FRAMEWORK

To properly test how selective rendering interacts with interactive ray tracing we modify a state-of-the-art interactive ray tracer. Manta [SBB+06] from the University of Utah was chosen for its up to date algorithms, optimised acceleration structures [Wal07], the packetised nature of its rendering pipeline and its extensibility due to the fact it is an open source project.

A detailed breakdown of the Manta pipeline is beyond the scope of this paper but a detailed overview is given in Stephens et al. [SBB+06]. For our experiment we modified the pipeline at the stage where it would have minimal impact on the performance of the rest of the pipeline. At this stage the pixels that need to be traced are specified and then converted into ray packets. We then provided a stride parameter which enabled us to simulate selective rendering by subsampling the image in a pre-defined way.

Stride in the context of this experiment is a parameter that defines how the selective rendering is performed. While simply rendering a lower resolution image would produce the same amount of pixels as utilising a stride this would not truly simulate selective rendering. This is because while selective rendering only renders a subset of the pixels in the image, the image is still created at full resolution. This has implications at many stages of the rendering pipeline and in order to produce a superior simulation it was decided to render the image at full resolution with a stride instead of just rendering the image at progressively lower resolutions. The stride itself simply defines how many steps one must take away from the current pixel before a new one is marked for processing. A stride of one simply means move over to the next pixel and mark it for processing, while a stride of two is move over twice, skipping one pixel and marking every second one for processing. Therefore a stride of X would render a $1/X^2$ of the total pixels in the image simulating the incoherent nature of selective rendering but allowing precise control over the amount of subsampling that took place. A ray is then generated per pixel and grouped into packets that are then traced through the scene. Figure 1 provides a graphical example.

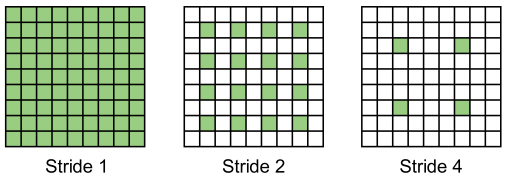
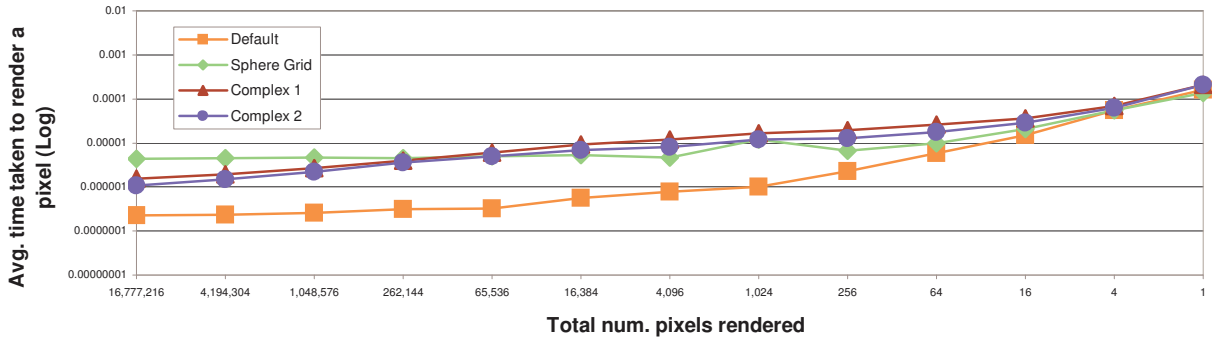


Figure 1: Changes in stride

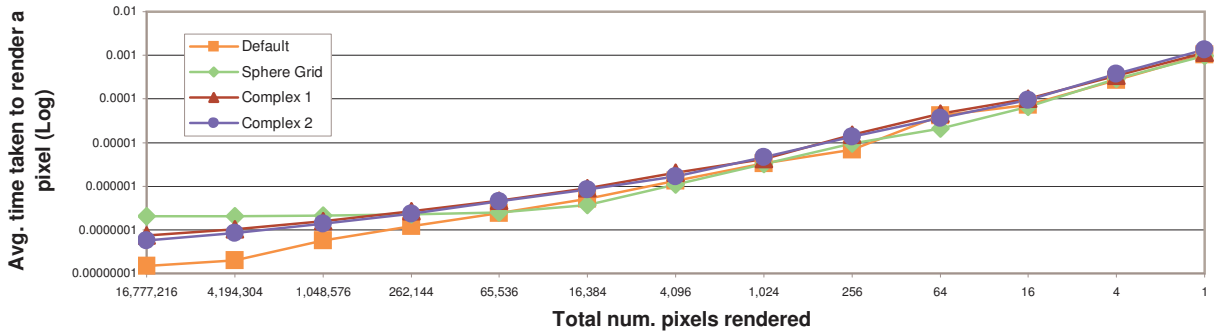
Once the selective pixel sampler was added to Manta four test scenes were selected (these are detailed in Section 4) and the experiment was setup in such a way to minimise unintentional cache coherence. This could occur through the re-use of the same scene while testing multiple strides where it was possible that some data may have remained in cache from the previous run and therefore would skew results. All four test scenes were run after each other for each of the strides to ensure that the cache contained data from a previous scene and not data from the same scene that could be reused.

A resolution of 4096×4096 to simulate a scene of 1024×1024 with 16 rays per pixel, was selected and each scene was run with thirteen different strides, from 1 to 4096, doubling the stride each time. This produced a range of images where every pixel in the image was being rendered to only one pixel being rendered.

For the experiment the standard Manta default of 64 rays per packet was used. Therefore for any results where less than 64 rays are processed the ray packet in Manta is not going to be be filled. This will lead to an increased cost per ray as the shading and other functions performed on the ray packet are no longer amortised over all 64 rays.

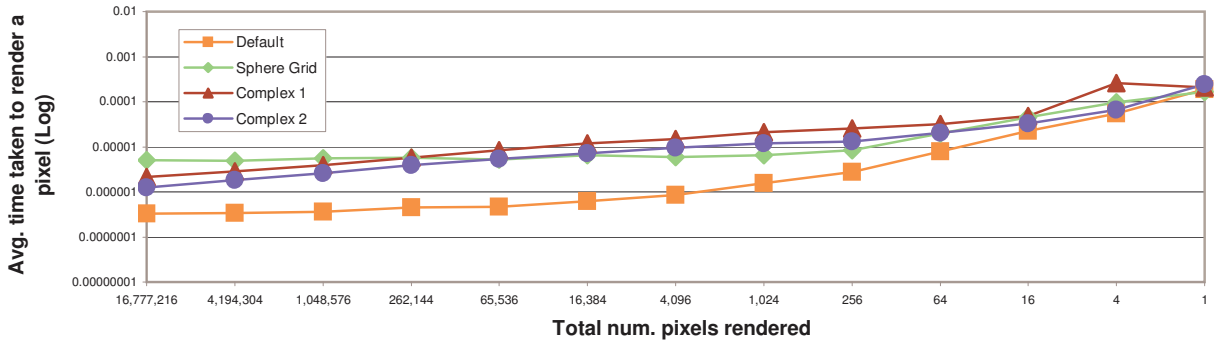


(a) Single-threaded

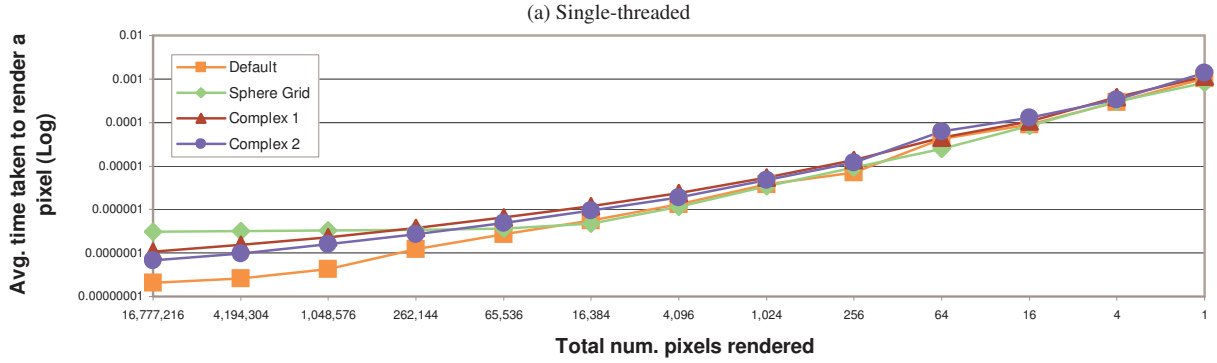


(b) Multi-threaded (8 threads)

Figure 2: Primary rays only

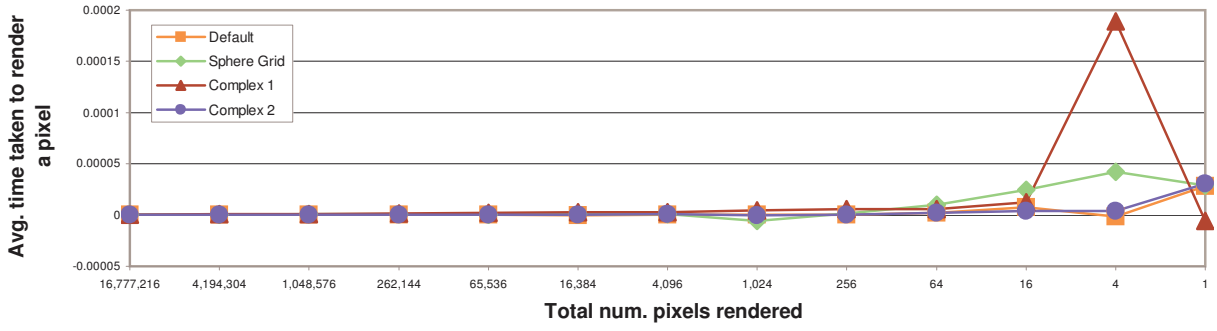


(a) Single-threaded

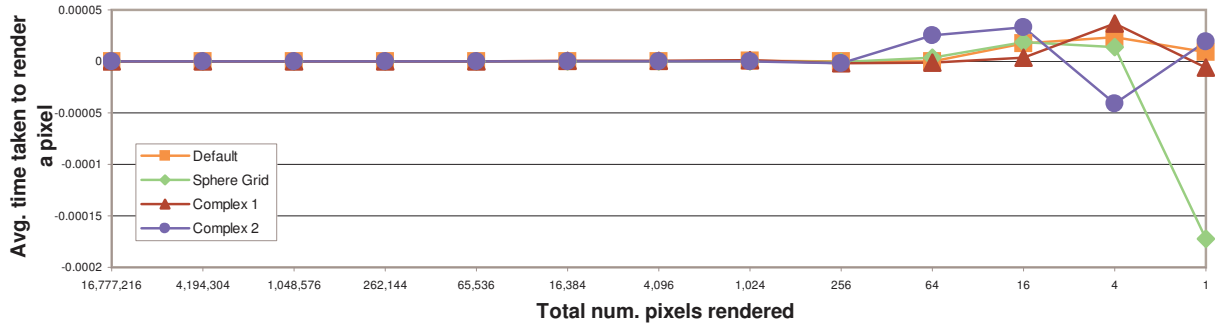


(b) Multi-threaded (8 threads)

Figure 3: Primary and secondary rays

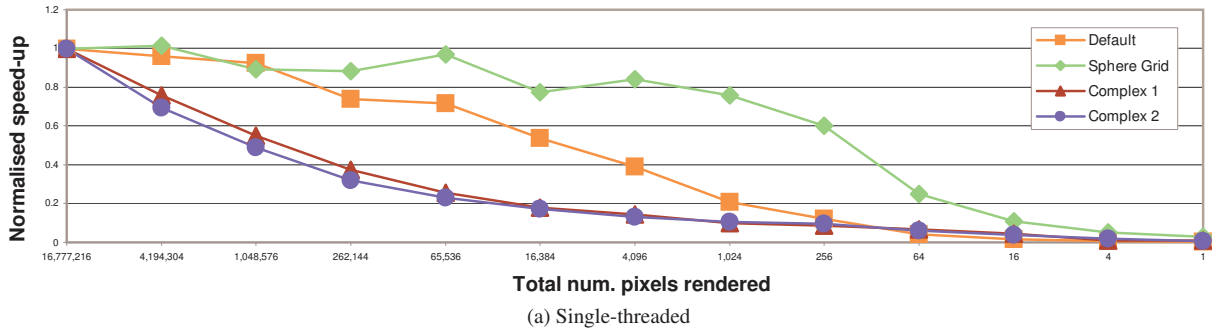


(a) Single-threaded

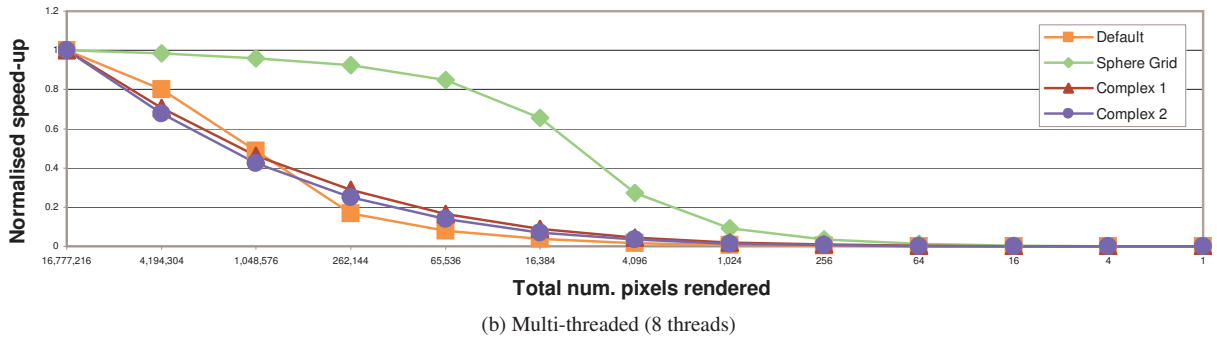


(b) Multi-threaded (8 threads)

Figure 4: Secondary rays only



(a) Single-threaded

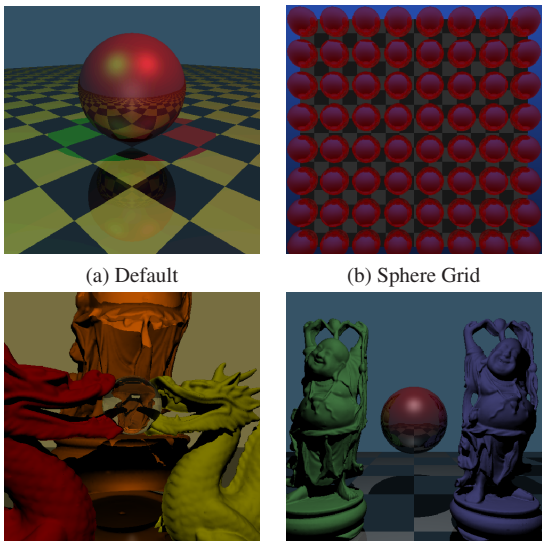


(b) Multi-threaded (8 threads)

Figure 5: Normalised speed-up compared to 4096×4096

4 RESULTS

We present our results for two cases using a single-core and an 8 core machine. All single-threaded tests were performed on an an single-core 2.8Ghz Intel Prescott CPU with two gigabytes of DDR memory running Linux with the 2.6.20-16 low-latency kernel. A modified version of Manta (detailed in Section 3) was compiled on the machine and run using one thread. The multi-threaded tests were run on an dual quad-core Intel 3.0 GHz XEON Apple Pro with four gigabytes of 667MHz DDR2 fully buffered ECC RAM. Manta was compiled under OSX 10.4 using the Apple GCC 4.0 (based on GNU GCC 4.0.1) and run with 8 threads.



(a) Default (b) Sphere Grid

(c) Complex 1 (d) Complex 2

Figure 6: The four scenes used

	Objects	Materials	Lights
Default	*2*	*2*	*3*
Sphere Grid	*65*	*2*	*1*
Complex 1	*698,274*	*5*	*3*
Complex 2	*586,466*	*5*	*2*

Table 1: Scene details

The results are summarised in 4 figures. Figures 2, 3 and 4 each display timings for specific components of each pixel. Figure 5 displays normalised speed-up for each stride.

Figure 2 and 3 show results for primary rays only and primary and secondary rays respectively. The data displayed on the Y axis is the average time taken to render a pixel using a logarithmic scale versus the total number of pixels rendered on the X axis using an exponential scale. Figure 4 contains data for secondary rays only, where the Y axis is the average time taken to render a pixel using a linear scale versus the total number of pixels rendered on the X axis using an exponential scale. Figure 5 shows the normalised speed-up for different strides when rendering both primary and secondary rays where the Y axis is the normalised speed-up using a linear scale versus the total number of pixels rendered on the X axis using an exponential scale.

One can see in Figures 2a and 3a that for the single-threaded results there is an overall increase in the average time taken to render a pixel as the total number of pixels rendered decreases, although it is not strictly a logarithmic increase and for certain scenes such as *SPHERE GRID* and *DEFAULT* is linear in certain areas. Examining the multi-threaded results one can see much less disparity between the different scenes and an almost perfectly logarithmic increase in *average time taken to render a pixel* as *total number of pixels rendered* decreases. This can most likely be attributed to the increased amount of cache, 16MB of L2 cache on the dual Quad-core vs. 2MB on the single-core Prescott, and as cache coherency decreases we see a higher overall cost per pixel, than was noted with the single-threaded results, for each ray.

The results in Figure 5 indicate, for a given stride, how close the results are to an ideal speed-up with the results being normalised to make comparison easier. An ideal speed-up being where the amount of time taken to render the given pixels is $T_1/(S_n^2)$ where T_1 is the time taken to render the image at a stride of 1 and S_n is the stride. For all scenes one can observe a decrease in speed-up as the stride is increased, and for all scenes other than *SPHERE GRID* the speed-up is only 20% of the ideal when the stride reaches thirty two. *SPHERE GRID* shows a much slower decrease in speed-up mostly due to its regular nature and higher coherence when calling shading and intersection routines. For both *DEFAULT* and *SPHERE GRID* we see that the decrease in speed-up is much faster in the multi-threaded results (Figure 5b) and while not as pronounced both *COMPLEX 1* and *COMPLEX 2* also show a decrease in speed-up when comparing the single-threaded and multi-threaded results. This, like the other results, shows that an increase in the amount of cache available adversely effects the performance of selective rendering.

What is also important to note is that due to the packet size being 64 rays, as described in Section 3, results where less than 64 pixels were processed showed an increase in average time taken per pixel, as ray packets where not completely filled when ray tracing occurred.

5 DISCUSSION

The implications of the results as pertaining to the interaction of selective rendering and interactive ray tracing are very interesting. As can be seen from Figure 2, 3 and 4 for the majority of scenes as the total number of pixels computed decreases the average time taken to compute a pixel increases logarithmically. One can

see an order of magnitude increase in the average time taken to compute a pixel as one goes from a stride of one to a stride of thirty two, the majority of this increase can be attributed to a decrease in cache coherency as the width of the packet increases with the growing stride.

Further evidence that this occurs as a result of cache misses and poor spatial coherency can be seen in the results of Figures 2 and 4. In Figure 2 the highly coherent primary rays are timed separately and show a constant and logarithmic increase in the average time taken to compute a pixel as the total number of pixels rendered decreases due to the stride increase. Only specular secondary rays are timed in Figure 4 and show that as the total number of pixels rendered decreases due to increased stride. There is very little measurable change in the average time taken to compute a pixel until only 256 samples are being calculated out of a possible 16,777,216. At this level any variance can be attributed to coincidental coherence or just not traversing complex parts of the scene. This shows that secondary rays are playing a very insignificant part in the overall deterioration of speed and that it is the very spatially coherent primary rays that are responsible for this speed loss. The loss in spatial coherence between the primary rays as stride increases is being directly translated into a loss of cache coherency and therefore an overall increase in average time taken to render a pixel. This is most apparent in multi-threaded results (Figure 2b) as the amount of L2 cache available to the CPUs increase.

As all the current interactive ray tracing techniques [SBB+06, RSH05] and optimised accelerating structures [SSK07, Wal07] rely on spatially coherent rays to amortise the cost of tracing large packets and when this spatial coherency is no longer present, as is the case with selective ray tracing, many of the speedups gained from these techniques are lost. On the other hand, secondary rays are naturally incoherent and due to this fact selective rendering has very little negative impact on the component of the ray tracing calculation as can be seen in Figure 4.

The discussion above along with results from Figure 5, which contain less than ideal speed-up that drops off sharply as stride increases for even the simplest scene, shows that selective rendering is useful but not as useful as it should be given the overall increase in average time taken to render a pixel. From this one can potentially draw two conclusions. One can either selectively render primary rays coherently using some form of adaptive or progressive approach or just not utilise selective rendering for primary rays and apply it just to secondary rays.

6 CONCLUSIONS AND FUTURE WORK

In this paper we have provided an analysis of the effects that selective rendering has on interactive ray tracing. We have modified an existing state of the art interactive ray tracer (Manta from the University of Utah [SBB+06]) to allow us to simulate different levels of selective rendering. Our results show that the average time taken to render a pixel increases as we increase the stride we render at and that this is mainly due to poor spatial coherence and its effects on primary rays. Primary rays rely on packets of spatially coherent rays for the traversal of acceleration structures and intersection of objects without this spatial performance deteriorates. This is problematic as cache coherence is heavily relied on by a large number of modern algorithms to provide the speed-up necessary for interactivity. We also see that secondary rays, when rendered with a stride, contribute almost nothing to the increase in average rendering time for a pixel, as opposed to primary rays. This is because secondary rays are naturally incoherent and aren't effected by poor spatial coherence and a lack of cache coherence.

From our results we conclude that while selective rendering in an interactive ray tracer can be useful the penalties incurred must be carefully managed. For future work we will look into designing selective rendering algorithms that carefully manage primary rays by making use of adaptive or progress methods that maintain spatial coherency. Furthermore, we will review selective rendering algorithms that perform selective calculations solely on the secondary rays.

ACKNOWLEDGEMENTS

This work was supported by the UK-EPSRC grant EP/D069874/2. We would like to thank the University of Utah for making Manta openly available.

REFERENCES

[BM98] Mark R. Bolin and Gary W. Meyer. A perceptually based adaptive sampling algorithm. In *SIGGRAPH '98*, pages 299–309, New York, NY, USA, 1998. ACM Press.

[CCW03] K. Cater, A. Chalmers, and G. Ward. Detail to attention: exploiting visual tasks for selective rendering. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 270–280, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[CCWG88] Michael F. Cohen, Shenchang Eric Chen, John R. Wallace, and Donald P. Greenberg. A progressive refinement approach to fast radiosity image generation. In *SIGGRAPH '88*, pages 75–84, New York, NY, USA, 1988. ACM Press.

[Cla76] James H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[CPC84] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *SIGGRAPH '84*, pages 137–145, New York, NY, USA, 1984. ACM Press.

[Deb06] Kurt Debattista. *Selective Rendering for High Fidelity Graphics*. PhD in Computer science, University of Bristol, 2006.

[FSPG97] James A. Ferwerda, Peter Shirley, Sumanta N. Pattanaik, and Donald P. Greenberg. A model of visual masking for computer graphics. In *SIGGRAPH '97*, pages 143–152, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[HMYS01] Jörg Haber, Karol Myszkowski, Hitoshi Yamauchi, and Hans-Peter Seidel. Perceptually guided corrective splatting. In A. Chalmers and T.-M. Rhyne, editors, *EG 2001 Proceedings*, volume 20(3), pages 142–152. Blackwell Publishing, 2001.

[LWC+02] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.

[LYTM06] Christian Lauterbach, Sung-Eui Yoon, David Tuft, and Dinesh Manocha. RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. In *2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–45, 2006.

[Mit87] Don P. Mitchell. Generating antialiased images at low sampling densities. In *SIGGRAPH '87*, pages 65–72, New York, NY, USA, 1987. ACM Press.

[Muu95] M. J. Muuss. Towards real-time ray-tracing of combinatorial solid geometric models. In *Proceedings of BRL-CAD Symposium '95*, June 1995.

[Mys98] Karol Myszkowski. The visible differences predictor: Applications to global illumination problems. In *Rendering Techniques*, pages 223–236, 1998.

[PMS+99] Steven Parker, William Martin, Peter-Pike J. Sloan, Peter Shirley, Brian Smits, and Charles Hansen. Interactive Ray Tracing. In *1999 Symposium Interactive 3D Computer Graphics*, pages 119–126, 1999.

[Pri01] J. Prikryl. *Radiosity methods driven by human perception*. PhD thesis, Institute of Computer Graphics and Algorithms, Vienna University of Technology, 2001.

[PS89] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. In *SIGGRAPH '89*, pages 281–288, New York, NY, USA, 1989. ACM Press.

[RPG99] Mahesh Ramasubramanian, Sumanta N. Pattanaik, and Donald P. Greenberg. A perceptually based physical error metric for realistic image synthesis. In *SIGGRAPH '99*, pages 73–82, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[RSH05] Alexander Reshetov, Alexei Soupikov, and Jim Hurley. Multi-level ray tracing algorithm. *ACM Trans. Graph.*, 24(3):1176–1185, 2005.

[SBB+06] Abraham Stephens, Solomon Boulos, James Bigler, Ingo Wald, and Steven G Parker. An Application of Scalable Massive Model Interaction using Shared Memory Systems. In *Proceedings of the 2006 Eurographics Symposium on Parallel Graphics and Visualization*, pages 19–26, 2006.

[SCCD04] Veronica Sundstedt, Alan Chalmers, Kirsten Cater, and Kurt Debattista. Top-down visual attention for efficient rendering of task related scenes. In *VMV 2004 - Vision, Modelling and Visualization*. Stanford, November 2004.

[SSK07] Maxim Shevtsov, Alexei Soupikov, and Alexander Kapustin. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *ACM Trans. Graph.*, 26(3), 2007.

[SWS05] Jörg Schmittler Sven Woop and Philipp Slusallek. Rpu: A programmable ray processing unit for realtime ray tracing. In *Proceedings of ACM SIGGRAPH 2005*, July 2005.

[WABG06] Bruce Walter, Adam Arbree, Kavita Bala, and Donald P. Greenberg. Multidimensional lightcuts. *ACM Trans. Graph.*, 25(3):1081–1088, 2006.

[Wal07] Ingo Wald. On fast Construction of SAH based Bounding Volume Hierarchies. In *SIGGRAPH '07*, 2007.

[War94] Gregory 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, New York, NY, USA, 1994. ACM Press.

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26(1):6, 2007.

[WDG02] B. Walter, G. Drettakis, and D. Greenberg. Enhancing and optimizing the render cache, 2002.

[WDP99] Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In D. Lischinski and G.W. Larson, editors, *Rendering techniques '99*, volume 10, pages 235–246, June 1999.

[WFA+05] Bruce Walter, Sebastian Fernandez, Adam Arbree, Kavita Bala, Michael Donikian, and Donald P. Greenberg. Lightcuts: a scalable approach to illumination. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1098–1107. ACM Press, 2005.

[WH06] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). pages 61–69, September 2006.

[Whi80] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.

[WK06] C. Wächter and A. Keller. Instant Ray Tracing: The Bounding Interval Hierarchy. In T. Akenine-Möller and W. Heidrich, editors, *Rendering Techniques 2006 (Proc. of 17th Eurographics Symposium on Rendering)*, pages 139–149, 2006.

[WMG+07] Ingo Wald, William R Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G Parker, and Peter Shirley. State of the Art in Ray Tracing Animated Scenes. In *Eurographics 2007 State of the Art Reports*, 2007.

[WSBW01] Ingo Wald, Philip Slusallek, Carsten Benthin, and Markus Wagner. Interactive Rendering With Coherent Raytracing. In *EUROGRAPHICS 2001*, pages 153–164, Manchester, United Kingdom, September 2001.

[YPG01] Hector Yee, Sumanita Pattanaik, and Donald P. Greenberg. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Trans. Graph.*, 20(1):39–65, 2001.