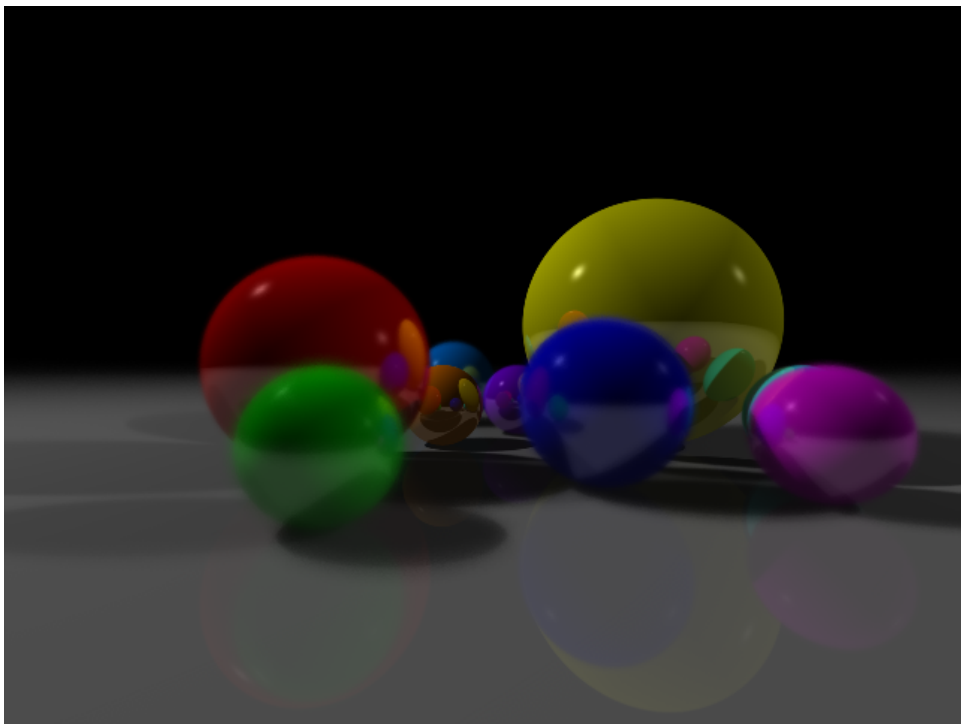
Research Methodology

Ray Tracing



Tom Farmer
(809130)

Contents

1 Introduction

Ray tracing is a simple method of producing realistic computer generated graphics. First described in its simplest form by Arthur Appel in 1968 (Appel, 1968), decades of research have developed ray tracing into one of the most effective means of generating high-quality CG images. The core principle of ray tracing is to cast rays from each pixel of the view plane into a scene, and determine the final colour of those pixels by detecting collisions between the rays and objects in the scene. Accompanying this report is a C++ implementation of a simple ray tracer, demonstrating some of the most common features of modern ray tracers. In an effort to determine where improvements in render time could be made, the impact that some of these features have on rendering time is evaluated in Section 4.

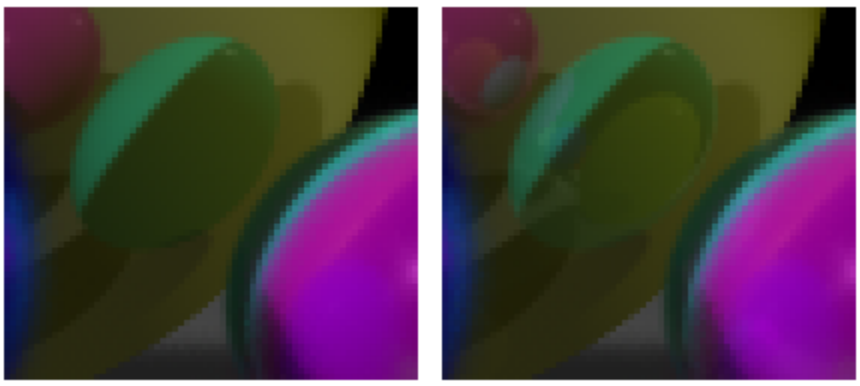
2 Features

2.1 Shadows

Appel's paper introduced a method of shading each pixel by casting a ray from each screen pixel into the scene. After the detecting the closest object in the scene intersected by the ray, a simple check is performed to determine if another object lies between the point of intersection and the light source. If the straight line from the light source to the point of intersection in interrupted by another object, that point is considered to be in shadow. Modern ray tracers generalise this principle to compute the contribution of an arbitrary number of lights, as does the implementation accompanying this report.

2.2 Reflections

Turner Whitted (1980) introduced recursive ray tracing, whereby each ray cast can be reflected or refracted if it intersects an object in the scene. Lighting contributions from subsequent intersections are added to the pixel's final colour. Although this model allows for realistic modelling of refraction, it is not implemented here. Each initial ray cast thus takes exactly one path through the scene and does not spawn more than one ray after each intersection. Determining how many reflections to compute for each pixel can be done by either specifying a maximum reflection depth at the start of execution or stopping when the contribution of each additional reflection becomes too small. The former method was chosen for this report because it simplifies performance testing by providing greater control over the rendering parameters. The effect of increased reflection depth is shown below:



2.3 Phong Lighting

Phong lighting (Phong, 1975) is a simple method of approximating realistic lighting on a surface by defining a material for each object in the scene. Materials have an ambient, diffuse and specular component, which specify the object's interaction with light.

2.4 HDR Rendering

Modern graphics hardware commonly uses 8-bits for each colour component (red, green and blue). The additive contribution to a pixel of lights in the scene is capped at 255, causing very bright areas to appear dimmer than they should. As a ray tracer making use of only a CPU, and not additional graphics hardware, the ray tracer accompanying this report uses 32-bit floating point numbers to represent each colour component to a high degree of accuracy. In HDR rendering, each component is clamped to the floating-point range 0-1 only after the contributions of all lights and reflections have been added, allowing very bright areas in reflections to correctly impact the colour of the pixel as a whole (Green and Cebenoyan, 2004).

2.5 Anti-Aliasing

Anti-aliasing is a collection of methods of smoothing the appearance of edges on screens with a finite number of pixels. Uniform grid-based super-sampling has been implemented here, which casts multiple rays from each pixel's physical limits on the view plane and averages their colours.

2.6 Depth of Field

Real-world cameras have an aperture (a hole though which light enters the camera) and a resulting focal length, limited the range of distance from the camera in which the world appears sharp. This phenomenon can be simulated by a ray tracer by casting additional rays from random points on a circular aperture on the view plane towards a pre-determined point on a focal plane (Cook et al, 1984).

2.7 Concurrency

The brute-force nature of ray-tracing lends itself to a parallel design (Culler et al., 1999, p.80). Pixels can be divided among available threads and rays cast independently of other pixels. Thread safety is a concern because all pixel colours must be written to the same buffer, and locking/unlocking objects introduces some performance overhead.

2.8 Dithering

Although each colour component is stored as a 32-bit floating point number, output to any modern monitor will limit the real colour range of each component to 0-255. Ffloyd-Steinberg dithering (Ffloyd and Steinberg, 1976) is therefore implemented as a post-processing effect to simulate the appearance of greater bit-depth and minimise visible colour banding.

3 Testing Methodology

Some of the features outlined above are of more interest in a performance comparison than others. In order to isolate the performance impact of a given feature, a set of default options was chosen and only the parameter of interest changed between each test. The default settings are as follows:

Setting	Value
Image width	640
Image height	480
Horizontal field of view	1.5708
CPU threads used	4
Maximum reflection depth	1
Anti-aliasing samples	2
Depth-of-field samples	100
Aperture	0.4
Focal depth	12
Dithering	ON

These settings can be used by executing the ray tracer with the following command:

```
raytracer -w 640 -h 480 -x 1.5708 -t 4 -r 1 -s 2 -d 100 -a 0.4 -f 12 -i 1
```

An example render using these settings can be seen on the title page. Note that the horizontal field of view is measured in radians, where 1.5708 radians is equal to 90 degrees. All tests were performed on a consumer desktop PC with an Intel 4670K processor running at 3.6GHz.

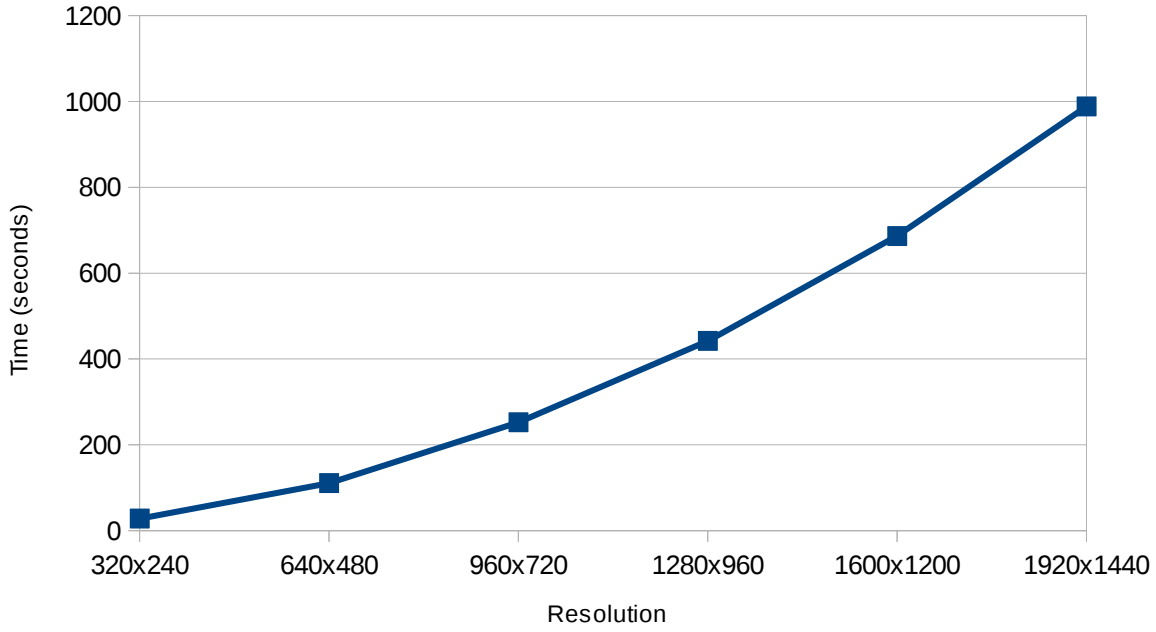
4 Performance Evaluation

Here the results of a series of tests are presented together with discussion of any apparent trends.

4.1 Resolution

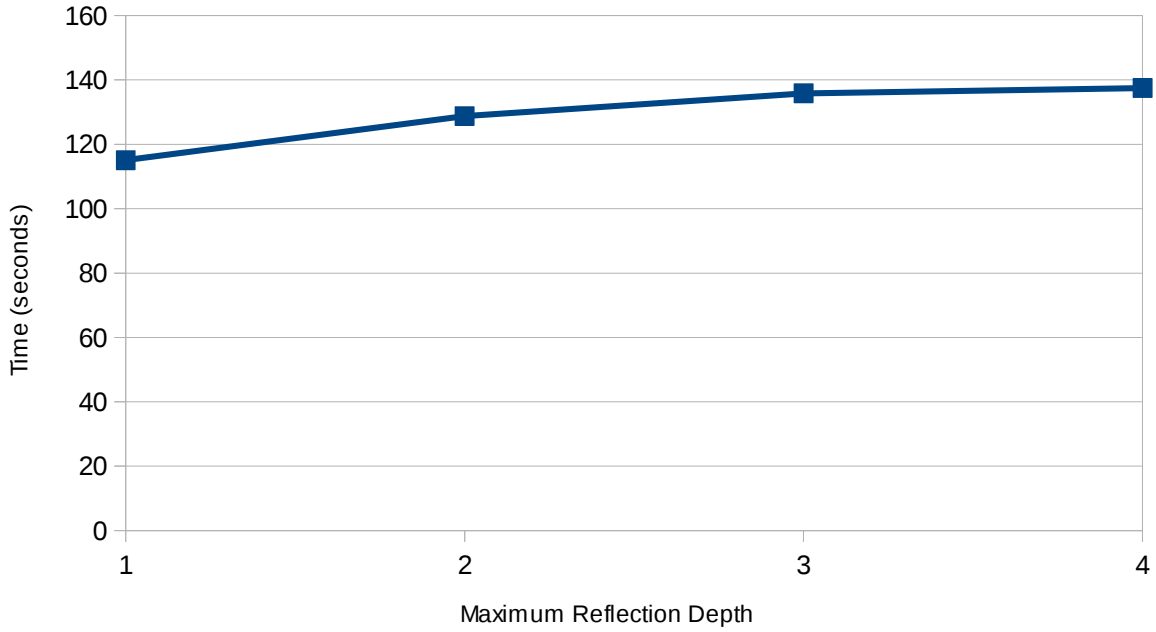
Resolution is, in theory, one of the major factors affecting the render time of a ray tracer. In more advanced implementations, optimisations are made to the way in which the scene is searched by each ray, reducing the time taken for each individual pixels. But even in these cases, the average cost of computing the colour at each pixel would be expected to be relatively constant as the resolution is increased. Thus, for any implementation of the fundamental ray tracing algorithm, one would expect the render time to have a positive linear relationship with the number of additional pixels added in each test.

The aspect ratio of the camera is another important factor to consider when testing the impact of resolution changes on render time. If the aspect ratio differs between tests, although the total number of pixels sampled may be equal, the distribution of the rays they produce in the scene will vary. For example, if two tests are conducted, the first using an aspect ratio of 4:3, the second using an aspect ratio of 16:9, each with a constant horizontal field of view and total pixel sample size, the second render will produce a wider image, effectively clipping the top and bottom in comparison to the first image. Rays cast into the scene may collide with a different set of objects, changing the render time unpredictably. A fixed aspect ratio of 4:3 was therefore chosen for these tests. Each test added 360 horizontal pixels and 240 vertical pixels to the final output image.



The results show a clear positive relationship between the total pixels sampled and the time taken to complete each render. A further analysis of the raw data (Appendix 6.1) reveals the cause of the gradual strengthening of this relationship. Each additional test adds the same number of pixels to the final output image on the x- and y-axes in a linear fashion, but the total pixels increases exponentially. This is reflected in the render time.

4.2 Reflection

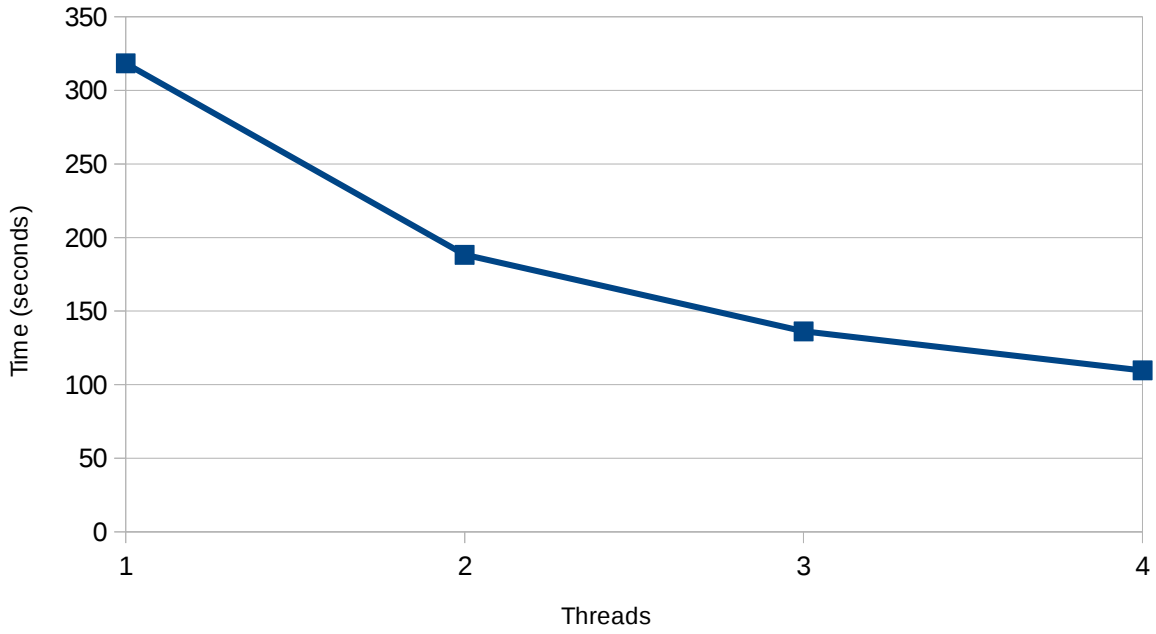


The data show a positive but weakening relationship between the maximum reflection depth and the total render time. The reduced increase in render time for each additional reflection appears to be caused by a decreasing number of reflection rays intersecting shapes after each reflection; some rays which reflect off a shape will not intersect with another shape. In the worst-case, a ray will be reflected as many times as the maximum reflection depth will allow, but in many cases a ray may, for example, reflect once and never intersect another shape in the scene. In statistical terms, if a given ray has a probability **p** of intersecting a shape, then the chance **q** of the ray reflecting **n** times is given by **q = np**. i.e. it decreases, causing the trend evident in the data. The strength of this effect with depend on the scene.

The gradual levelling-out of the curve results in highly comparable render times when the maximum reflection depth is set to three or four, suggesting a greater reflection depth could be used with little impact on performance, but with likewise little impact on visual quality as the number of pixels showing a reflection tends towards zero.

4.3 Concurrency

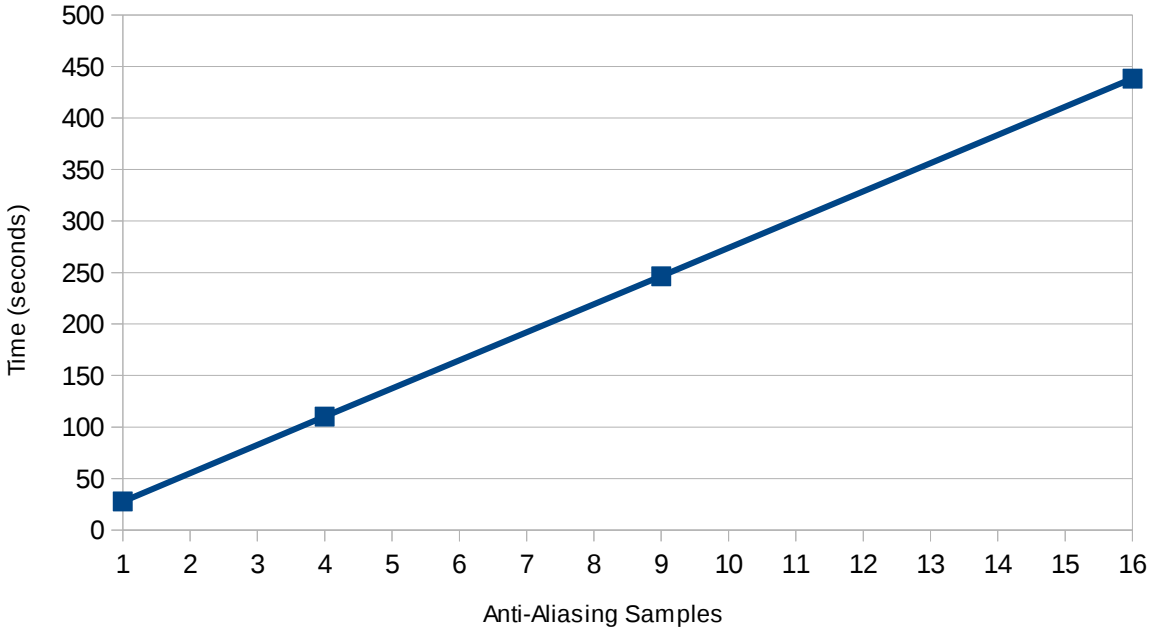
Ray tracing's brute force approach makes it highly appropriate for an implementation taking advantage of modern multi-core CPUs. Because these tests were conducted on a quad-core Intel CPU, up to four threads have been tested.



With the doubling of the number of available cores, one might expect to see a halving of render time. The data do not show this relationship. Instead, although there is a clear reduction in render time as each additional thread is used, the fall is not directly proportional; doubling the number of threads from one to two cut render time by only 40%. Likewise, four threads completed the operation in 42% less time than two. This could be due to overhead involved in ensuring that the program remains thread-safe, even when several threads are writing to the same render buffer.

4.4 Anti-Aliasing

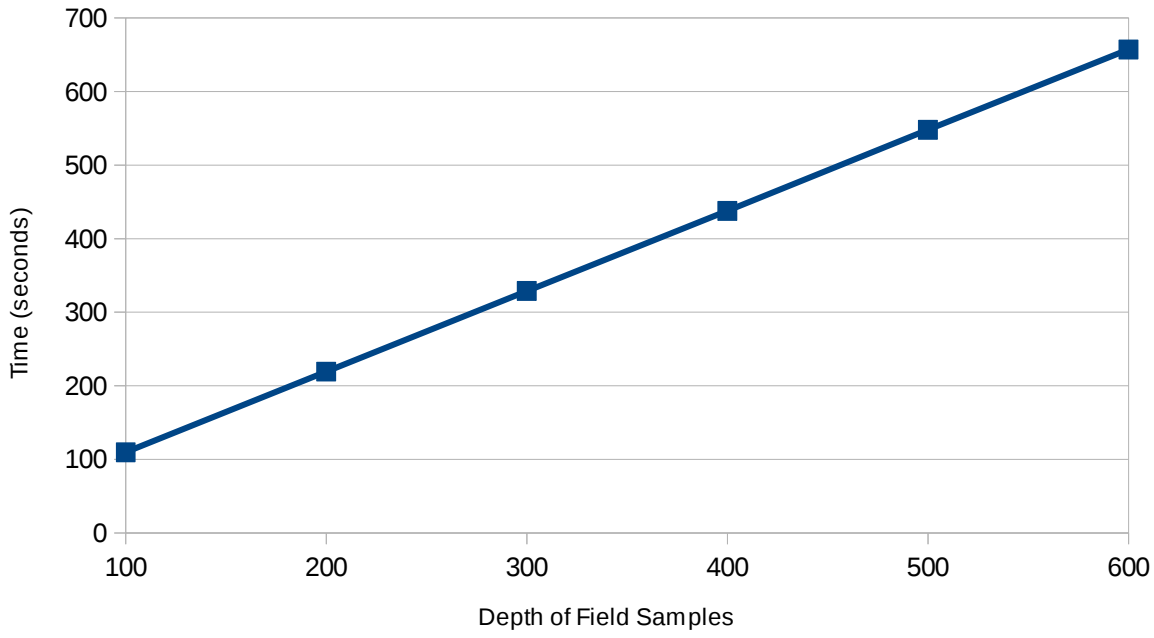
The super-sampling used in this implementation is non-adaptive, meaning that the number of samples is fixed and identical for every pixel. These samples are arranged evenly within the pixel's physical boundaries on the view plane in equal numbers of rows and columns. Total samples are therefore squares of the number of samples in each dimension.



Super-sampling casts a fixed multiple of the number of pixels into the scene within a small area. Additionally, more rays cast from similar origins in similar directions are more likely to take similar paths through the scene. One would therefore expect an increased number of rays cast into the scene to have a linear impact on the render time. This relationship is present in the data.

4.5 Depth of Field

Because super-sampling and depth of field are implemented in a similar way, one would expect both to have a similar per-sample impact on performance. Unfortunately, depth of field appears to necessitate the use of many more samples than super-sampling to achieve the desired effect.



The linear trend seen in the super-sampling tests is also present here, confirming the similar nature of the two techniques. The lower values in the sample range used in this test, such as 100 and 200 samples, produce an undesirable hazing effect in the final image. Only when using around 500 samples do these artefacts begin to disappear, leaving a realistic approximation of physical depth of field.

5 Conclusion

Ray tracers used in the graphics industry, such as those which render each frame of Disney Pixar's creations, need to be as efficient as possible to minimise render time and reduce production costs. The implementation presented here is inefficient by the standard of such ray tracers because it does not use advanced techniques to reduce the time needed to determine collisions with shapes in the scene. The time required to produce images at higher resolutions for example, already suggests an unsustainable rising trend, which would be worsened by the addition of more techniques to improve image quality (such as adaptive super-sampling and texture mapping). Some of the more advanced literature on this topic makes reference data structures such as octrees as a means of reducing render time, and would be worth considering in further work.

6 Appendix

6.1 Resolution

	320x240	**640x480**	**960x720**	**1280x960**	**1600x1200**	**1920x1440**
Time (seconds)	27.9366	110.72	252.464	441.961	686.409	988.226

6.2 Reflections

	1	**2**	**3**	**4**
Time (seconds)	115.08	128.756	135.835	137.495

6.3 Threads

	1	**2**	**3**	**4**
Time (seconds)	318.282	188.302	136.193	109.689

6.4 Anti-Aliasing Samples

	1	**4**	**9**	**16**
Time (seconds)	27.712	110.165	246.477	438.248

6.5 Depth-of-Field Samples

	100	**200**	**300**	**400**	**500**	**600**
Time (seconds)	109.652	219.244	328.824	437.763	547.872	657.132

7 Bibliography

Appel, A. (1968). “Some techniques for shading machine renderings of solids”, AFIPS '68 Conference Proceedings. pp. 37-45.

Cook, R. et al. (1984). “Distributed Ray Tracing”, SIGGRAPH '84 Proceedings of the 11th annual conference on Computer graphics and interactive techniques. pp. 127-145.

Culler, D. et al. (1999). “Parallel Computer Architecture: A Hardware/software Approach”, Morgan Kaufmann Publishers, Inc. (ISBN-13: 978-1-55860-343-1)

Ffloyd, R. and Steinberg, L. (1976). “An adaptive algorithm for spatial greyscale”, Proceedings of the Society of Information Display 17. pp. 75-77.

Foley, J. et al. (1996). “Computer graphics: principles and practice 2nd ed. in C”, Addison-Wesley Publishing Company, Inc. (ISBN: 0-201-84840-6)

Gillies, D. (2009). Interactive Computer Graphics lecture slides. Available online at:
http://www.doc.ic.ac.uk/~dfg/graphics/graphics2009/
[Accessed 1 December 2014]

Green, S. and Cebenoyan, C. (2004). “High Dynamic Range Rendering on the GeForce 6800”, NVIDIA. Available online at:
http://http.download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_HDR.pdf
[Accessed 10 December 2014]

Phong, B. (1975). “Illumination for computer generated pictures”, CACM, 18(6). pp. 311-317.

Whitted T. (1980). “An improved illumination model for shaded display”, CACM, 23(6). pp. 343-349.