# An Efficiency Comparison Between Deferred Shading and Forward Shading in Varying Complexity Scenes

Aldo Curtis

August 29, 2014

Word count:6631
GBHE0114.
16867.
Jason Colman.
AldoCurtis@gmail.com

## 1 Abstract

This project intends to compare Deferred and Forward shading in terms of their efficiency while rendering realtime scenes. Implementations of both rendering methods are used to render scenes with varying complexity, during which FPS and frametime was recorded. The results indicate that for complex scenes Deferred Shading performs more effectively and consistently then Forward Shading.

# Contents

## 2 Introduction

This project intends to quantitatively test the effectiveness of Deferred Shading and Forward Shading in scenes with different levels of complexity with regard to lighting and geometry. Both methods are still used in games, and both have advantages and disadvantages.

In 1993 one of the striking differences noted in the new video game Doom was the introduction of lighting as shading. Various video games before Doom simulated 3D environments with carefully created backgrounds, using shaded pixel art to give the illusion of depth. However these assets were always generated in advance before the game play. In Doom's predecessor Wolfenstein 3D, every level shared the same ambient light, and pools of light were built into billboard textures to imitate the light produced by ceiling lights. However, characters and other features were not affected by these lights as the programme does not dynamically light objects.

In its time Doom's comparatively simple lighting was described as remarkable : "truly scary" (Anon, 1994), and now dynamic, calculated lights are used on hundreds of games to create immersive and believable 3D worlds. With the rising demand for visual acuity in real time scenarios there is a constant quest for newer, faster and more efficient techniques to enhance what can be achieved in games. Detailed and dynamic lighting can be computationally expensive, depending on how it is implemented. Many modifications such as sector culling and baked shadows have to be employed in order to increase efficiency and optimize results.

Forward Shading is the most commonly used method for shading and lighting. For every visible object, it iterates through every light that could affect that object and performs the relevant lighting calculations. It then uses these results to calculate the final pixel colour.

Deferred Shading is a relatively new technique, only becoming possible to implement in recent years. It calculates lighting as a post process, after geometry rasterisation. Deferred Shading has many desirable algorithmic properties (Shishkovtsov, 2005), providing predictable performance (depending on lights in the screen area), perfect depth complexity and a reduction in the number of batches per frame. Compared to Forward Shading it simplifies the lighting complexity for a scene, rendering the number of lights independently from the number of objects.

However Deferred Shading requires large texture read/write memory bandwidth, and struggles with translucency and hardware assisted anti-aliasing. Because of this Forward Shading is still the preferred method offering simplicity in implementation and low computational overhead.

## 3 Background

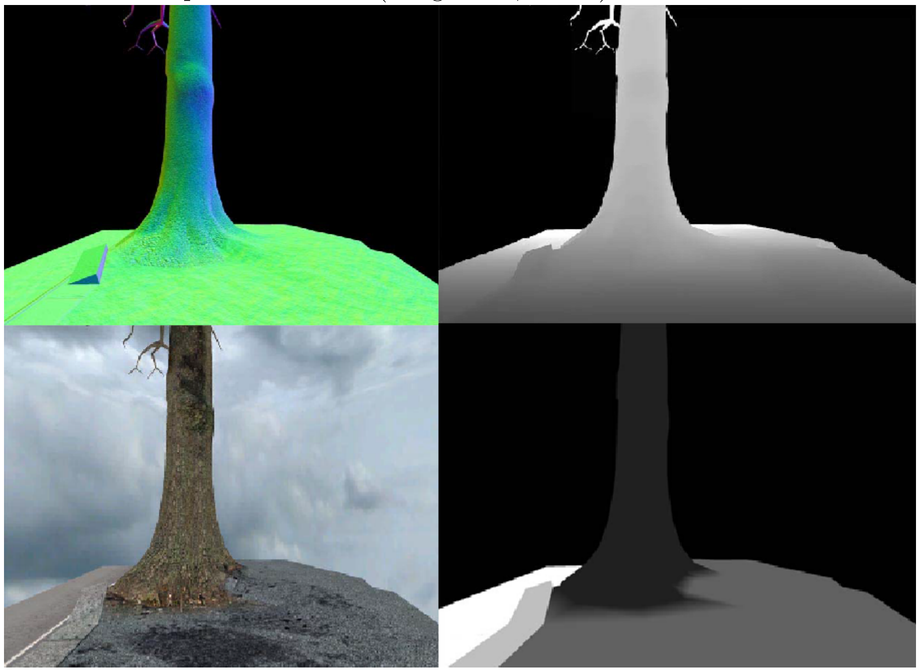
Lighting in computer graphics functions by artificially simulating the light refection of surface properties assigned to geometry in the scene. Surface properties normally include: a texture or image representing the colour of the surface, a specularity value, which denotes how reflective a surface is, and other variables used for differing graphical techniques. Like the surfaces, a number of properties are assigned to lights which are then used to calculate colour values produced when the light interacts with said surfaces. Typical properties would include ambient, diffuse and specular colours for global illumination, eye-position dependent illumination and specular highlights respectively (Phong, 1973).

Lighting can be a computationally expensive process. Dynamic lights and objects which can move around the scene under the control of player actions eliminate the possibility for surfaces to have lighting baked into their colours (pre-calculated); lighting calculations must be performed dynamically. Forward Shading performs a very simple operation for each light included in the scene. Firstly it checks to see if the light can affect a given surface. Initially this is done with a range check for point and cone lights and subsequently performs a reflection vector check, simulating the angle of vision and calculating whether the camera can perceive the reflected light as a diffuse or specular coloration. An issue with Forward Shading is that it must iterate through every light for every rendered surface, effectively causing the complexity to comprise the number of light multiplied by the number of surfaces (numOfLights * numOfSurfaces).

The concepts behind Deferred Shading have been used in many projects before its original inception as a graphics rasterizer (Tebbs, 1992). Surfaces that store material IDs, depth values and normals for the entire screen have been used in multiple projects and are present in todays depth buffers, where textures with the same dimensions as the screen store information per pixel and are read for graphical effects. The depth buffer stores depth values which can be used to prevent superimposition when rendering geometry in an arbitrary order.

Deferred Shading applies lighting as a post process rather than applying lighting while drawing geometry, simplifying the lighting calculations to the number of lights added to number of surfaces (numOfLights + numOfSurfaces). To do this, properties of the visible geometry are saved to textures sharing the same dimensions as the screen which holds properties necessary for lighting. These properties may vary between projects, but some remain consistent: world position, specularity and normal values.

Figure 1:A picture showing the various types of information stored in the GBuffer if interpreted as colour.(Hargreaves, 2004b)



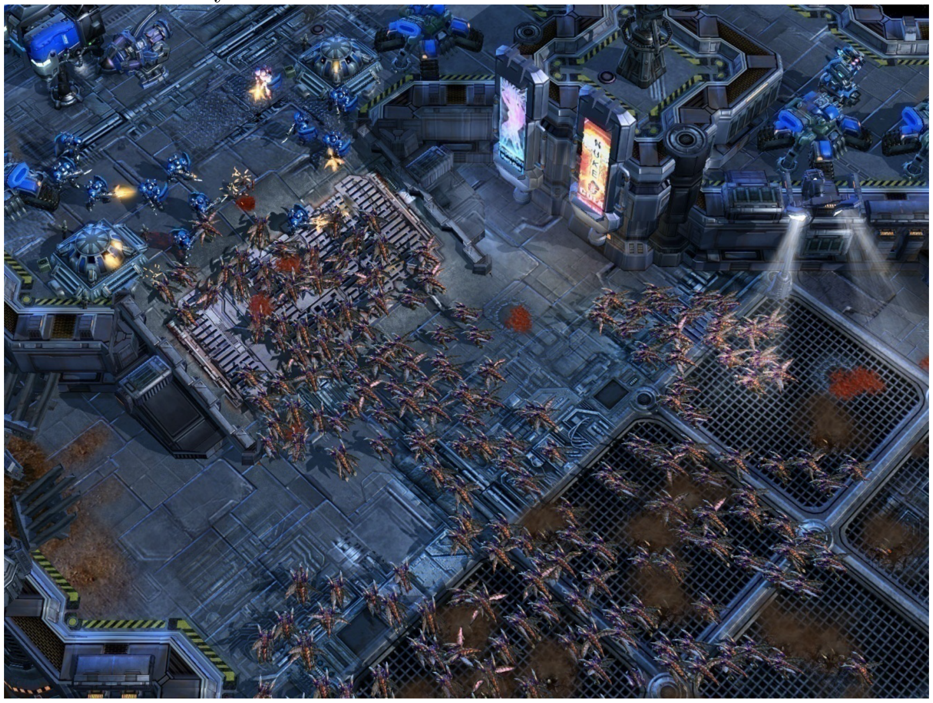
The first paper introducing Deferred Shading was published in 1988 (Deering, 1988), it originally discussed circuit diagrams that could be used to construct hardware to achieve the technique. This was before configurable rendering pipelines and shaders, and because of this had to be constructed at the circuit level effectively specialising the hardware for this single technique. The name Deferred Shading was coined at a later date, and was not mentioned in the original paper.

Deferred Shading was first widely introduced in 2004 (Hargreaves, 2004a), when a presentation at Evolve (a games programming conference in San Fransisco) gave an insight into implementing it on current hardware using shaders and multiple render targets(MRT). MRT allowed a fragment shader to return multiple values to different textures in one rendering pass. However, each texture bound to the MRT must have the same resolution, and same encoding format, which can be an inconvenience.

Games with high geometric complexities and possibilities for moderate to large overdraw can benefit from the use of Deferred Shading. Overdraw occurs when any number of polygons are shaded without being rendered to a final pixel on the screen. This is normally due to multiple draw calls, when a subsequently rendered object occludes the first, and replaces the product of its shading. Overdraw can lead to many more shading calculations than necessary. The properties of Deferred Shading makes it (depending on hardware) much faster for large numbers of lights to be rendered and increases scalability and possible complexity of scenes at a preferable frame-rate. (Shishkovtsov, 2005) (Filion, 2008a) (Lee, 2009)

Deferred Shading treats lights very similarly to geometric objects, assigning them bounding volumes that can be used to test whether the light is close enough to an object to effect it. These bounding volumes can be culled in much the same way that normal geometry can be culled, using optimizations such as Frustum Culling, game level zoning and dynamic level of detail operations to reduce the number of lights. (Hargreaves, 2004a)(Harris, 2004)

Figure 2: A visual example of what is considered as a complex scene. Starcrafts use of Deferred shading allows dynamic lighting and light emitters to interact with every unit.



Other techniques such as screen space ambient occlusion were also successfully implemented in Deferred Shading engines. In this technique, an ambient occlusion value is stored in the GBuffer to reduce the amount of ambient light that an object receives due to occluding geometry. (Filion, 2008a)

The main disadvantage of Deferred Shading is that, for many games, the memory storage and throughput requirements necessary to sample and save a GBuffer of such size are impractical. Initially there was no way to represent translucency, or take advantage of hardware assisted anti-aliasing. Graphics cards now usually offer efficient means of anti-aliasing. However, Deferred Shading cannot take advantage of this because all geometry is rendered to the GBuffer rather than directly to the framebuffer. In this case, anti-aliasing could be counter productive, as it would distort values in the depth buffer which do not refer to colour (such as normal values). Additionally, MRT does not currently support anti-aliasing. (Engel, 2013) (Hargreaves, 2004a)

There are methods of allowing hardware assisted antialiasing to be used with Deferred Shading. One example is Decoupled Deferred Shading(Engel, 2013).This method stores information about the polygons being rendered independently from their surface information. With storage of visibility data in a buffer they can use antialiasing on the slimmer visibility buffer without significant texture throughput requirements.

For Deferred Shading to include translucency more complex scene management systems had to be created. These involve changing the order in which objects are drawn and blending colours from translucent objects into the GBuffer.

These disadvantages have not stopped some games and game engines adopting Deferred Shading as the main rendering method (Andersson, 2009). Further methods to improve the Deferred Shading technique are being tested at the time of writing this project, including hybrid Deferred Shading/lighting engines, use of compute shaders as light accumulators and screen space ambient occlusion using the GBuffer. (Filion, 2008a)

## 4 Implementation

This project creates a testing environment in which both Forward Shading and Deferred Shading can be implemented. As it is the industry standard, Forward Shading is very well supported in libraries such as DirectX and OpenGL. This means that it requires very little additional programming to be implemented. However, Deferred Shading has only become practical in the last few years, so is not nearly as well supported. Therefore, in order to implement it, there were a number of additional minimum technical requirements: multiple render targets (MRT), a stencil buffer, depth buffer control and blending.

The GBuffer is composed of four screen sized textures which will store the necessary values for Deferred Shading. To write to the GBuffer efficiently, MRT is required so that the fragment shader can output multiple values to the GBuffer simultaneously in one shader execution. MRT was initially not supported by DirectX or OpenGL but became part of the DirectX framework in DX9, and OpenGL 2.0.

The stencil buffer is a necessary as part of a technique to test whether the light bounding volumes that have been rasterised intersect with the stored geometry from the GBuffer in an efficient manner.

Depth buffer control is used to disable the depth test when using techniques that do not require it. In addition, it is also used in memory optimisation deployed to reconstruct the 3D world position of an object from a stored depth value. Blending is needed to keep lights independent of one another. Each light contributes its illumination to the final frame buffer by using the blending operations to add its colour to the final frame buffer without having to interact with any other lights being rendered.

Most of these features are typical in graphical rendering programmes, and are integrated into both OpenGL and DirectX.

Direct X11 was chosen for this project because it is a widely used rendering platform which most programmers are familiar with. A small Direct X framework was used to simplify common rendering tools such as textures, primitives, and *obj* mesh loading.

Simple Phong shading was used for lighting calculations. Phong is a interpolated lighting method, using normals to calculate the colour of light given its reflection angle(Phong, 1973). It was chosen to give desirable visual results.

## 4.1 Forward Shading

Forward Shading was implemented by iterating through every light in the fragment shader. The lighting calculations are accumulated in the fragment shader and returned as a total. A light buffer, bound as a constant buffer (DirectX 11 implementation), stores all of the relevant lights for the scene and is bound as a shader resource for use in a fragment shader. Similarly, the material of the surface being rendered is stored as a texture. Two main optimisations were implemented for the Forward Shading part of this project. A maximum range representing an area of influence around point lights was implemented. If the geometry being shaded is outside of this range then no further calculations will be performed for this light. This range check is done using squared distance to avoid excessive square root operations. Secondly, there are no branching if statements to allow efficient optimisation of the shader code.

In both Deferred and Forward Shading directional lights share the same complexity, as every pixel in the screen is affected. In Forward Shading, this simply involves applying these directional lights to every object rendered. In Deferred Shading, this is represented as a full screen quad with no stencil intersection tests.

## 4.2 Deferred Shading

The rendering method is split into two stages. The first stage deals with the rasterisation of geometry while the second stage explains the stencil technique used to render light bounding volumes and the final pixel shading.

### 4.2.1 Stage 1:GBuffer Creation

The first stage of the rendering process for Deferred Shading is GBuffer creation. The GBuffer consists of 4 RGBA 16bit textures, which is typical for Deferred Shading. (Harris, 2004) (Lee, 2009)

The shading technique used in this project is a simple one, and does not require all of the textures and their channels to be used for storing values. More complex techniques will require more texture usage. For example, screen space ambient occlusion would require an ambient occlusion value in the GBuffer format. The variables shown here could be considered the absolute minimum necessary to perform Phong shading, but this format is larger than necessary as one of the textures is never used. This serves simulate to texture requirements similar to a more complex project.

The chosen formats were as follows:

| Texture. | RGB | A |
|---|---|---|
| 1 | Normal values | Screen space depth |
| 2 | Diffuse colour | Specular value |
| 3 | Specular colour | N/A |
| 4 | N/A | N/A |

The values in the GBuffer represent properties of rendered geometry in the scene. Each pixel in the GBuffer relates to the visible geometry being transformed to the location in screen space of that pixel. In this format the position of the geometry is not explicitly stored, but is calculable from the provided depth value.

This process backward projects 2D co-ordinates on the screen into the 3D world. Using the projection matrix, it is possible to create a ray which indicates the direction that an object occupies in order for it to be represented by that pixel on the 2D screen. Using the depth value it is possible to calculate where on that ray the object was, and recover its 3D world position. This is performed in the lighting stage per pixel shaded in order to recover their 3D position for lighting calculations.

In the event that there are unused channels in the GBuffer, it may be more efficient to store the position in its XYZ co-ordinates in order to simplify the calculations needed in the second rendering stage. As games tend to use all of the channels available this is rarely applicable. This project intends to simulate that of a more complex project implementation, so will use this more memory efficient technique. This technique could be even more memory efficient if it were possible to sample the depth buffer in the fragment shader, as the value is already stored there and would not need to be stored in the GBuffer.

This first stage requires both vertex and fragment shader execution on every object in the scene. Using MRT it writes to each texture using the channels to store the data collected. Diffuse colour is sampled from the relevant surface texture and stored in its RGB components as well as the normal vector of the face stored using XYZ. However the fragment shader writes values to the GBuffer instead of the frame buffer. Rendering straight to the frame buffer occurs in Forward Shading.

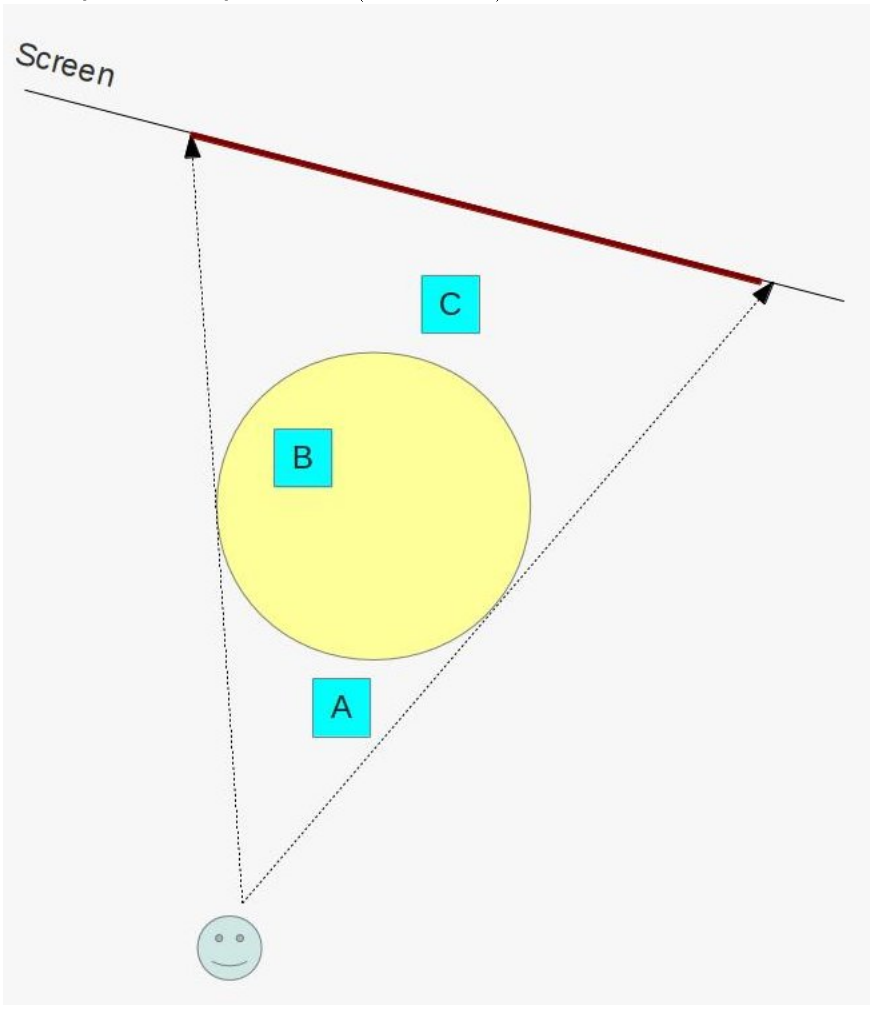
#### 4.2.2 Stage 2:Shading

Unlike Forward Shading, Deferred Shading treats lights very similarly to objects. Every light type requires a bounding volume represented by geometry. For point lights the bounding volume is a sphere with its centre at the location of the point light, and a radius equal to the range of the point light. For lights that affect all objects in a scene a quadrilateral covering the entire screen can be used. The bounding volume is rendered similarly to an object. However, the shader is only executed on areas of intersection between the light bounding volume and objects saved in the depth buffer.

Each light is rendered in two stages. The first light stage prepares the stencil buffer for use in a technique that determines if depth values stored in the GBuffer intersect the light bounding volumes. The stencil buffer is a utility screen sized texture which can be used in many different ways for different techniques. While rendering the light bounding volumes the program records each time a face fails the depth test, indicating that it is behind something in the depth buffer. For this technique we disable back face culling and draw all faces of the bounding volume.

While drawing the object, we decrement the value in the stencil buffer if the front face fails the depth test, indicating that for that pixel something is in front of the light. We also increment the value in the stencil buffer if the depth test fails for the back faces, indicating that something is in front of them. If the depth value in the GBuffer indicates that an object is in front of the back faces of the bounding volume but behind the front faces of the bounding volume, it is clear that the object is intersecting the bounding volume so should be shaded by the light. This is indicated by a stencil buffer value of 1. This method can also function with atypical geometry and multiple objects, where non 0 values indicate the geometry is intersecting at least one of the lights.

Figure 3:An example of the stencil buffer technique used to detect collisions with light bounding volumes. (Meiri, 2014)



Seen in Figure 2, the face represents the camera position looking into the scene, the yellow sphere is the light's bounding volume and the three labelled squares are objects in the scene. When the light volume is rasterized, depth tests around square A will both increment and decrement the stencil value, as both the front and back faces of the sphere are occluded behind the object; leaving it at zero. C will neither increment nor decrement as both faces pass the depth test. Object B however will increment the stencil values of the back faces behind it, but will not decrement them, indicating that it is intersecting the light bounding geometry.

The above technique requires a shader to execute, but should not write to either the depth or frame buffer. However, it must be able to read depth values. This means that, when using this technique, the programme prevents writes to both the frame and depth buffer. Instead, a null shader, returning no value in the fragment shader, was used. This method allows stencil operations to execute without changing anything else.

The second stage of the lighting process uses the stencil values written in stage one to reduce the number of pixels that need to be shaded. The stencil test is an early rejection operation which is performed on each pixel. In this case, only values not equal to 0 are accepted, otherwise the stencil test cancels all further calculations on that pixel including depth dests and shading calculations. This process reduces the number of calculations needed per light significantly as the shader will only execute on areas that the light is affecting, and unlike forward shading will not have to test light ranges for each pixel.

In this implementation, front face culling is enabled. This is so that if the camera is inside the light volume being rendered, only the back faces of the light bounding volume will be rendered, as the front faces will have been culled. In the event that the camera is inside a light volume being rendered, no front faces will be rendered as none are visible so back faces must be rendered for the light volume to be properly rasterized. Depth testing must be disabled, as now the visible faces of the bounding volume will be behind the object they are intersecting. This is true by definition as to intersect the volume the object must be closer to the camera than the back faces of the light bounding volume.

Lighting calculations are performed on each pixel, taking into account the properties of the light, and the saved properties of the surface at that pixel which can be extracted from the GBuffer. These values are then blended into the final frame buffer using the addition blend operation. The addition operation simply adds two colours together, providing the behaviour of light accumulation, integrating the individual colours returned by each lighting calculation into the final colour.

This project consists of a simple optimization which avoids clearing the stencil buffer for further lights. In the second stage the stencil pass operation resets the stencil value to 0, and as the stencil test requires values that are not equal to 0 it will reset the stencil buffer while performing test operations.

## 5 Hypothesis

The aim of this project is to compare the efficiencies of Forward and Deferred Shading when rendering complex scenes. In this context, complex refers to scenarios with numerous lights and objects. The hypothesis is that Deferred Shading will perform better then Forward Shading for scenes that are more complex, but might perform less well when rendering simpler scenes.

Deferred Shading simplifies lighting complexity by rendering lights independently of both the scene geometry and other lights. For this reason, large numbers of lights can be rendered much faster. However, if texture memory is limited then Deferred Shading may perform less than optimally because of the large memory requirements for the GBuffer.

These tests aim to discover if using current hardware, simple Deferred Shading implementations can be more efficient than classical Forward Shading methods. It proceeds to explore in what scenarios the two methods work optimally.

The expectation is that Forward Shading, due to its low computational overhead, is faster with simple scenes and small numbers of lights, whereas Deferred Shading will run faster in complex scenes. Deferred Shading should run more consistently across the different scenes whereas the performance of Forward Shading should fluctuate more. This is because Forward Shading has unpredictable complexity for each frame, with the addition of each light multiplying the number of lighting calculations needed per object. Deferred Shading however will only have to render one additional light in its lighting stage, regardless of the number of objects.

Where there is one global light, both techniques share the same complexity. However, Deferred Shading will have more overhead costs for such a simple scenario. This is because the light must affect all objects in the scene, and therefore lighting calculations must be performed on all pixels regardless of techniques used to reduce complexity.

## 6 Methodology

To ensure similar test environments all non critical processes were closed before testing and the program is run for the same duration for each test. Each test was performed three times and the values averaged over the three trials.

The purpose of these tests is to compare the efficiency of both techniques when rendering scenes with large numbers of lights. This test is intended for real time scenarios, so Frames Per Second (FPS) and Frametime are the variables that were measured. These variables were chosen as they represent how well the application runs in real time, FPS giving a direct runtime efficiency value and frametime representing the computational complexity of the frame as it must have taken so many micro seconds to execute. The tests were performed on a laptop so heat dissipation for the graphics card is considered. In order to compensate for this, five minute breaks were taken between each test. The camera rotated around a scene for sixty seconds to observe it from all angles. A ten second wait time before rendering starts was included, both to set up the scene and to attach the analytics programme to the process.

Two tests were performed. The first test consisted of scenes with a static number of lights differing in complexity. This was to test the basic performance of the techniques in these situations. The second test consisted of a static scene with gradually increasing lights and a rotating camera.

The purpose of the last tests is to provide information about how the two methods fare under dynamic rendering conditions. The tests also indicate how the two methods perform with incrementally increasing complexity providing more precise data.

Three scenes of varying geometric complexity were used. The first contained one model and was the simplest. The second scene contained two models which rotate around cardinal axes and two subdivided icosahedrons. The third scene comprised of a hundred spheres and four rotating models.

For the first test, both methods were used for each scene. All measurements were performed with 160 lights.

The second and third tests used the most complex scene (scene 3) and started with no lights, gradually increasing them at a rate of 10 per second up to a total of 590.

To avoid over saturation each light was given a squared distance attenuation and no more than 0.3 in each colour channel. The attenuation value means that the lights effects decreased substantially with range, and the low colour values ensured that the buildup of light on each object did not white out" the lighting on surfaces. Whiting out occurs when the amount of light affecting a surface increases the total accumulated light past 1 in any colour channel. This is not possible in real lighting scenarios but in this case could cause graphical detriment to rendering.

These lights were placed and coloured using a pseudo-random number generation system. Every scene was generated using the same seed (the priming number for pseudo-random number generation) providing exactly the same light configuration for consistency. The seed number is 1337 (arbitrarily chosen), using the srand() function of the C++ maths library. The use of this seed would recreate the same conditions on every computer making re-testing possible.

There is a limit to the number of lights that can be included using the Forward Shading method. For every pixel rendered, every light that can effect that pixel must be available in a constant buffer. Constant buffers have a limited size, and there can only be a limited number of them bound to the shader at any one time. This project does not approach this limit, but it is worth mentioning that Deferred Shading does not have this problem, as it renders every object and light independent of one another.

The variables that were tested were average FPS and frametime per second. Averaging these variables per number of frames would have given skewed results because when the techniques performed differently the results would have a different number of total frames after completion. Therefore, averages were calculated per second of execution.

Because of Deferred Shading's algorithmic properties, but large texture sampling overhead, it is expected it to perform less well than Forward Shading when dealing with small scenes, but its performance will stay more consistent with larger scenes whereas Forward Shading will decrease in performance.

The computer used is a laptop and its specs are as follows: Processor : Intel Core i7 Quad Core Mobile Processor i7-4710MQ (2.50GHz) 6MB RAM : 8GB Kingston SODIMM DDR3 1600Mhz. Graphics Card: NVIDIA GeForce GTX 870M.

The Intel Graphics Performance Analyzer suite (GPA suite) was chosen for analysis purposes. It is suitable because of its wide range of tools and accurate results. The tool offerers individual frame history analysis useful for debugging, testing and allowing each pixel to have its history sampled, displaying how the final output colour was reached.

The GPA suite also allows for direct output of recorded performance to CSV (Comma Separated Values) files, which permit convenient processing of data.

## 7 Results

The results were as follows, with the FPS and frametime averaged per second of execution. As mentioned above, this was to avoid the problems with higher frame rate samples having more points of data.

### 7.1 Experiment 1

**Table 1. Scene 1 FPS**

| Test Number | Deferred FPS average *fps* | Forward FPS average *fps* |
|---|---|---|
| 1 | 46.107 | 58.156 |
| 2 | 46.087 | 54.963 |
| 3 | 46.121 | 55.625 |
| Average Total | 46.105 | 56.248 |
| Average Standard Deviation | ±5.80 | ±12.695 |

**Table 2. Scene 1 Frametime**

| Test Number | Deferred FT average $\mu s$ | Forward FT average $\mu s$ |
|---|---|---|
| 1 | 22057.5 | 17735.9 |
| 2 | 22079.2 | 19239.3 |
| 3 | 22043.9 | 18768.2 |
| Average Total | 22060.2$\mu$s | 18581.1$\mu$s |
| Average Standard Deviation | ±2997.0 | ±3749.7 |

**Table 3. Scene 2 FPS**

| Test Number | Deferred FPS average *fps* | Forward FPS average *fps* |
|---|---|---|
| 1 | 30.086 | 28.268 |
| 2 | 29.670 | 27.832 |
| 3 | 31.577 | 28.017 |
| Average Total | 30.444 | 28.039 |
| Average Standard Deviation | ±6.623 | ±4.751 |

**Table 4. Scene 2 Frametime**

| Test Number | Deferred FT average $\mu s$ | Forward FT average $\mu s$ |
|---|---|---|
| 1 | 35628.4 | 37099.4 |
| 2 | 37176.8 | 37668.9 |
| 3 | 34814.1 | 37420.7 |
| Average Total | 35891.1$\mu$s | 37396.3$\mu$s |
| Average Standard Deviation | ±9868.4$\mu$s | ±7767.9$\mu$s |

**Table 5. Scene 3 FPS**

| Test Number | Deferred FPS average *fps* | Forward FPS average *fps* |
|---|---|---|
| 1 | 44.996 | 29.352 |
| 2 | 44.364 | 30.496 |
| 3 | 45.371 | 30.651 |
| Average Total | 44.910 | 30.166 |
| Average Standard Deviation | ±4.911 | ±7.134 |

**Table 6. Scene 3 Frametime**

| Test Number | Deferred FT average $\mu s$ | Forward FT average $\mu s$ |
|---|---|---|
| 1 | 22483.2 | 35776.4 |
| 2 | 22852.1 | 34721.7 |
| 3 | 22294.2 | 34542.4 |
| Average Total | 22546.5 | 35013.5 |
| Average Standard Deviation | ±2610.6 | ±7782.6 |

## 7.2 Experiment 2

Figure 1. FPS over time for Deferred and Forward shading.

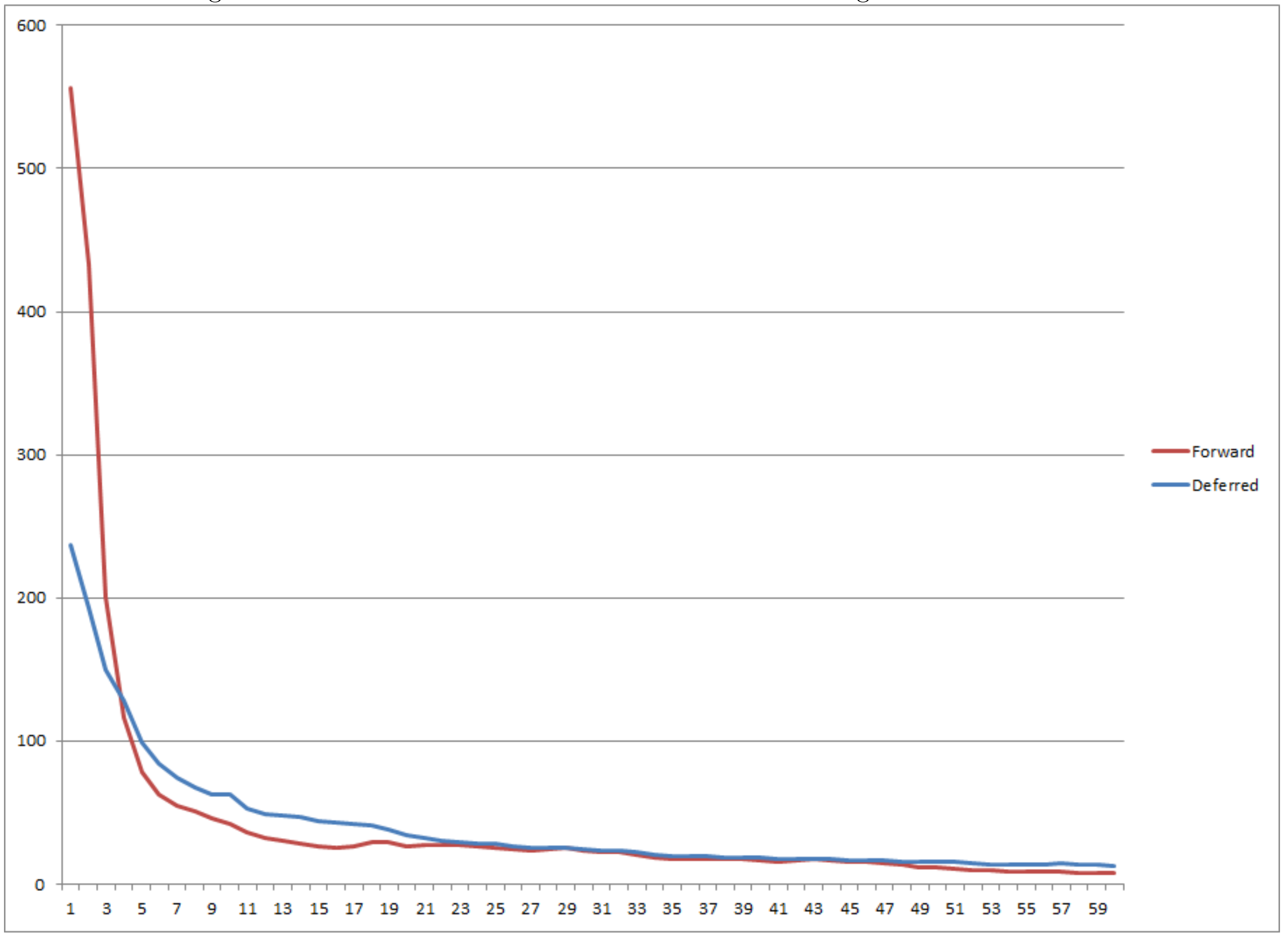
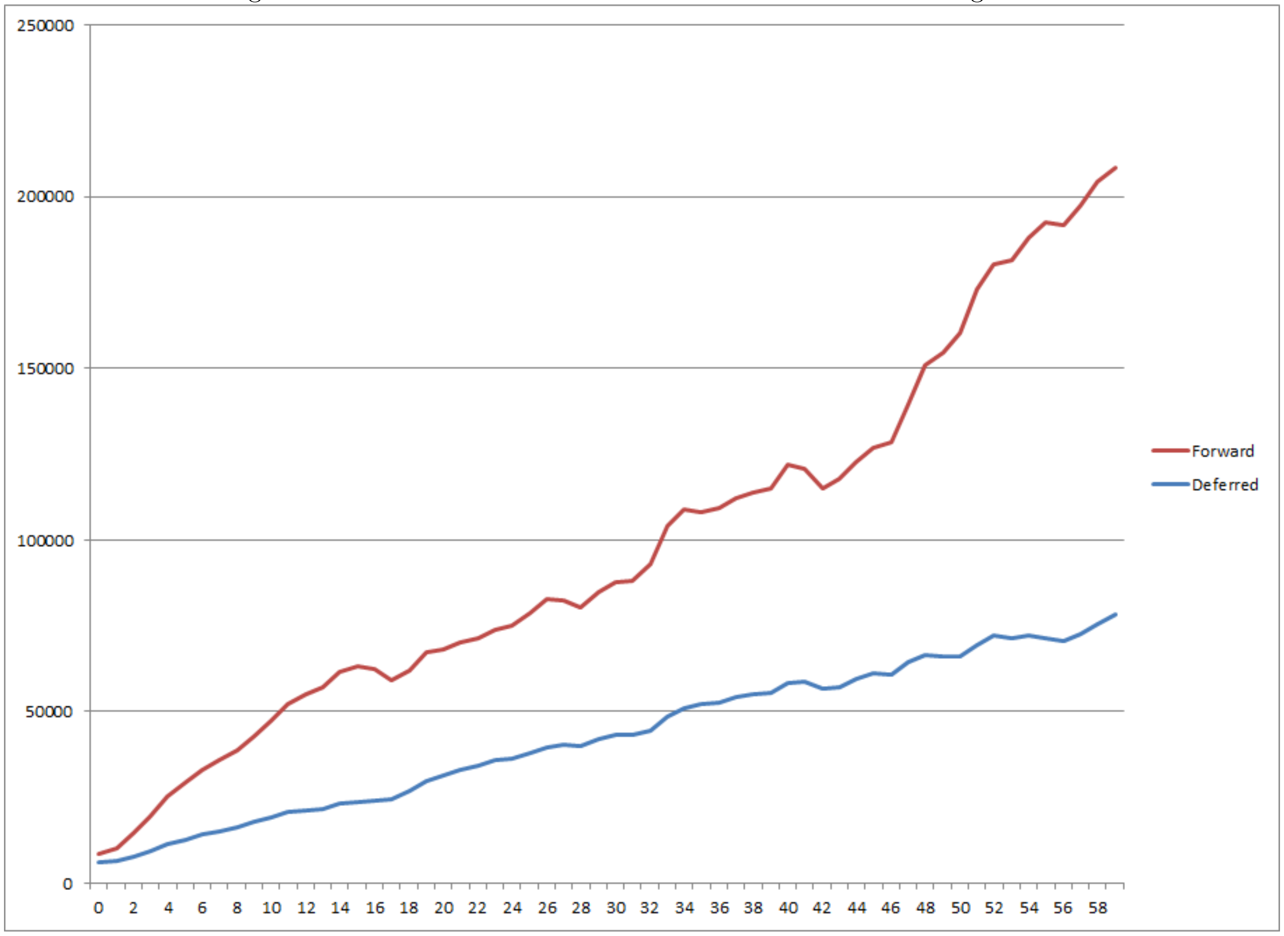


Figure 2. Frametime over time for Deferred and Forward shading.



## 8 Analysis

### 8.1 Experiment 1

#### 8.1.1 Scene 1

As illustrated in Tables 1 and 2, Forward Shading outperforms Deferred Shading for Scene 1 by an average of +10 FPS and an average difference of -3479.1s of frame time. This supports the theory that the high overhead of Deferred Shading will result in underperformance in comparison to Forward Shadings low overhead when rendering simple scenes.

However, Deferred Shading had a significantly smaller deviation in its FPS and frametime per second than Forward Shading. Deferred Shading showed a 5.8 FPS deviation compared to Forward Shading's 12.6, and a frametime deviation of 2997s compared to 3749s. While Deferred Shading did not perform as well on average, its performance was much more consistent than Forward Shading. In the Forward Shading test, the frame rate varied significantly more than Deferred Shading, affected by a changing camera angle and the visibility of lit objects. In this scene a single model rotated in the centre of the frame. The framerate in the Forward Shading test varied according to how much of the screen the model occupied depending to the angle at which it was being viewed.

### 8.1.2 Scene 2

Deferred Shading performed better than Forward Shading in this test, as seen in Tables 3 and 4. It's FPS was 2.3 frames faster on average, and the average frametime was 2500s lower. However, the difference between the two methods is far lower than one standard deviation (for both measurements), so there was no significant difference in performance.

The more interesting figure for this scene are the standard deviations. Both methods show high standard deviation, specifically noticeable in Deferred Shadings frametime, where the standard deviation of 9868s is a 27% deviance from the average frametime of 35891s.

It is noticeable in the results (as seen in the Appendix) that the framerate and frametime of Deferred Shading only decreased significantly once when the screen was entirely occluded. Both methods reached the same lower limit of around 19FPS. This represents a larger drop in performance for Deferred Shading than Forward Shading.

Both shaders performed poorly on Scene 2 compared to the other two scenes. Scene 2 was expected to be simpler to render than Scene 3, so the FPS was expected to be higher and the frametime lower. However, during the camera's rotation through Scene 2 there were a few seconds of complete occlusion, as one of the icosahedrons fills the screen. At this point both rendering methods struggled to maintain their framerates.

There are several possible reasons for this dramatic performance drop. Unlike the other scenes, the full occlusion of the screen would have required most operations to encompass the entire screen. to be continued...

In scenes 1 and 3, Deferred Shading maintained a low standard deviation (2997s and 2610s respectively), showing that the shader was performing consistently within each test. However, in Scene 2, the standard deviation is much higher (9868s), due to the full screen occlusion (as mentioned above). However, Forward Shading does not display the same pattern. Here, the standard deviation was similar in both scenes 2 and 3 (7767s and 7782s respectively), while comparatively much lower in scene 1 (3749.7s). This suggests that when using Forward Shading, an increase in geometric complexity significantly decreases the reliability of performance (thus increasing the standard deviation). This effect could be considered comparable to the effect of a full screen occlusion when using Deferred Shading.

#### 8.1.3 Scene 3

In scene 3, as seen in seen in Tables 5 and 6, Deferred Shading was on average 14FPS and 2350s faster then Forward Shading. Additionally it is important to note that the standard deviation for Deferred Shading stayed low at +-4.9FPS, and +-2610s, with similar values to the results from scene 1. This supports the hypothesis that Deferred Shadings overhead will become less significant when its efficiency with multiple lights becomes more relevant.

The standard deviation in this scene suggests that Deferred Shading maintains constant render times independently of scene complexity. The results from the normal scene seem to indicate that it suffers similarly to Forward Shading in the event of full screen occlusion. From these results, it is impossible to conclude whether this is a result of extensive overdraw, or having to render more pixels, and further testing would be required to understand this more completely.

### 8.2 Experiment 2

In the second experiment results were also collected and averaged per second. Figure 1 shows the dramatic drop in framerate of both techniques from their initial state of zero light. Forward Shading has an FPS of almost triple that of Deferred Shading at the beginning (with zero lights), which supports the hypothesis that Deferred Shading is comparatively inefficient when rendering simple scenes.

Approximately 3 to 4 seconds into the test (30-40 lights) there is a point at which Deferred Shading starts to outperform Forward Shading. Deferred Shading maintains its higher frame rate consistently from that point on. In the most extreme case (600 lights), Deferred Shading achieves an average of 11 FPS which is much faster than the 7 FPS achieved when using Forward Shading.

The frametimes also support the hypothesis of this document; that Deferred Shading is more effective when rendering scenes with complex lights. Figure 2 shows that the frametimes when using Deferred Shading increase almost linearly. However, when using Forward Shading, the frametime starts to increase dramatically at around 48 seconds (480 lights). If this trend continued with larger light counts, it is plausible to predict that the difference in performance will be even more noticeable.

## 9 Conclusions

In conclusion there is reasonable evidence to support the hypothesis that Deferred Shading performs better then Forward Shading for rendering complex scenes. The significance of this is demonstrable with as low as 4 object and 160 lights. Deferred Shading also maintains more consistent render times in all scenes, except for the circumstances of full screen occlusion shown in Scene 2.

For more complex and aesthetically elaborate games Deferred Shading seems to be a viable choice of rendering method, providing both consistency and greater rendering speeds with numerous lights. Implementation time and limitation on lower end hardware may cause problems with the Deferred Shading technique, but as hardware improves and developers adopt the technique it could replace Forward Shading in the foreseeable future.

In addition it is interesting to note that the maximum light limitations due to constant buffer memory constraints are not present when using Deferred Shading. However, on less powerful hardware, there is a possibility that the texture requirements may not be as negligible as with the graphics card these tests were performed on. However after overcoming this graphical requirement, Deferred Shading has minimal further requirements.

Reflecting on the project, there were many things that could have been done to improve the reliability and usefulness of the data. Designing the scenes to be more consistent with one another would produce more reliable results. The data collected using the second scene suffered in reliability and relevance because of the extra work needed to render and light pixels for the entire screen compared to that of the other scenes. Tests involving gradually increasing lights would be more accurate if the number of lights increased more slowly with more control over the lights positions. It could also have benefited from the use of ambient lights.

The simple lighting implementation used in this project could be improved by making it more similar to lighting implementations of finished realtime products, giving more valuable insight into the performances of both techniques under conditions imposed by the industry. Examples of possible changes include: zone culling for lights, allowing for multiple textures per object, screen space ambient occlusion and the possibility of fog and other rendering effects. While this data alone may be able to quantify its performance in the scenarios provided it is not adequate to reliably test both methods for all cases.

Texture memory bandwidth and bottlenecking were considered as variables to be measured. On the hardware tested it quickly became apparent that this simple implementation was not going to cause a bottleneck and was safely within the operational limits of the graphics card. However, these would be very useful variables if the same test was to be performed on less powerful hardware or consoles.

After finishing the tests it became clear that a number of other other variables, conditions and situations could be examined, all within the scope of comparing Forward and Deferred Shading. These were either too complex to implement in the time frame available, or not totally relevant to the comparisons made in this project. In order to develop the scope of this project, testing on different hardware types, player controlled cameras, outdoor and indoor designed scenes could all be viable parameters for fully exploring and determining the differences between these two techniques.

Deferred Shading is a viable rendering method for the future of computer graphics, making more complicated and detailed scenes possible for real time scenarios. There is a further possibility that hardware may become more optimised if Deferred Shading becomes more popular. This will increases its efficiency and availability as the successor to the Forward Shading method.

# 10 Appendix

## 10.1 Development Diary

## 10.2 Experiment 2 results

These results are averaged from the results of three tests performed for each method.

| Time | Deferred FT | Deferred FPS | Forward FT | Forward FPS |
|---|---|---|---|---|
| 0 | 6014.4 $\mu$s | 236.5 fps | 2627.9 $\mu$s | 556.4 fps |
| 1 | 6461.0 $\mu$s | 192.7 fps | 3640.3 $\mu$s | 432.4 fps |
| 2 | 7913.2 $\mu$s | 149.4 fps | 6596.9 $\mu$s | 200.5 fps |
| 3 | 9385.6 $\mu$s | 128.4 fps | 10166.2 $\mu$s | 116.0 fps |
| 4 | 11235.8 $\mu$s | 98.4 fps | 13838.2 $\mu$s | 77.8 fps |
| 5 | 12703.9 $\mu$s | 84.3 fps | 16522.7 $\mu$s | 63.0 fps |
| 6 | 14092.8 $\mu$s | 74.1 fps | 18923.8 $\mu$s | 54.9 fps |
| 7 | 15250.5 $\mu$s | 67.4 fps | 20567.7 $\mu$s | 50.4 fps |
| 8 | 16176.0 $\mu$s | 63.0 fps | 22488.8 $\mu$s | 46.3 fps |
| 9 | 17839.5 $\mu$s | 62.1 fps | 24918.1 $\mu$s | 41.5 fps |
| 10 | 19151.1 $\mu$s | 53.1 fps | 28045.6 $\mu$s | 36.4 fps |
| 11 | 20811.3 $\mu$s | 48.9 fps | 31214.7 $\mu$s | 32.5 fps |
| 12 | 21341.1 $\mu$s | 47.7 fps | 33527.5 $\mu$s | 30.2 fps |
| 13 | 21532.3 $\mu$s | 47.1 fps | 35439.2 $\mu$s | 28.4 fps |
| 14 | 23115.1 $\mu$s | 44.1 fps | 38319.3 $\mu$s | 26.4 fps |
| 15 | 23781.9 $\mu$s | 42.7 fps | 39274.5 $\mu$s | 25.7 fps |
| 16 | 24230.8 $\mu$s | 42.0 fps | 38054.9 $\mu$s | 26.4 fps |
| 17 | 24506.8 $\mu$s | 41.5 fps | 34509.0 $\mu$s | 29.2 fps |
| 18 | 26740.1 $\mu$s | 37.9 fps | 35200.1 $\mu$s | 28.9 fps |
| 19 | 29591.0 $\mu$s | 34.2 fps | 37905.0 $\mu$s | 26.7 fps |
| 20 | 31238.8 $\mu$s | 32.3 fps | 36959.1 $\mu$s | 27.4 fps |
| 21 | 33141.8 $\mu$s | 30.5 fps | 37136.9 $\mu$s | 27.3 fps |
| 22 | 34118.1 $\mu$s | 29.5 fps | 37283.5 $\mu$s | 27.0 fps |
| 23 | 35730.7 $\mu$s | 28.2 fps | 38064.3 $\mu$s | 26.4 fps |
| 24 | 36246.8 $\mu$s | 27.8 fps | 38936.1 $\mu$s | 25.9 fps |
| 25 | 37992.4 $\mu$s | 26.5 fps | 40704.7 $\mu$s | 24.8 fps |
| 26 | 39589.9 $\mu$s | 25.5 fps | 43146.9 $\mu$s | 23.4 fps |
| 27 | 40313.5 $\mu$s | 25.0 fps | 41888.8 $\mu$s | 24.1 fps |
| 28 | 40122.9 $\mu$s | 25.1 fps | 40423.8 $\mu$s | 24.9 fps |
| 29 | 41794.4 $\mu$s | 24.1 fps | 42856.5 $\mu$s | 23.6 fps |
| 30 | 43217.8 $\mu$s | 23.3 fps | 44287.0 $\mu$s | 22.8 fps |
| 31 | 43333.8 $\mu$s | 23.3 fps | 44651.2 $\mu$s | 22.6 fps |
| 32 | 44273.5 $\mu$s | 22.7 fps | 48570.7 $\mu$s | 20.8 fps |
| 33 | 48496.1 $\mu$s | 20.7 fps | 55393.4 $\mu$s | 18.1 fps |
| 34 | 51150.2 $\mu$s | 19.7 fps | 57636.0 $\mu$s | 17.4 fps |
| 35 | 52052.8 $\mu$s | 19.3 fps | 55904.5 $\mu$s | 18.0 fps |
| 36 | 52737.0 $\mu$s | 19.1 fps | 56546.0 $\mu$s | 17.7 fps |
| 37 | 54303.2 $\mu$s | 18.5 fps | 57774.2 $\mu$s | 17.4 fps |
| 38 | 54926.3 $\mu$s | 18.3 fps | 58753.6 $\mu$s | 17.1 fps |
| 39 | 55319.7 $\mu$s | 18.2 fps | 59600.3 $\mu$s | 16.8 fps |
| 40 | 58458.9 $\mu$s | 17.2 fps | 63319.5 $\mu$s | 15.8 fps |
| 41 | 58789.0 $\mu$s | 17.1 fps | 62143.7 $\mu$s | 16.1 fps |
| 42 | 56659.1 $\mu$s | 17.8 fps | 58560.7 $\mu$s | 17.1 fps |
| 43 | 56993.3 $\mu$s | 17.6 fps | 60736.1 $\mu$s | 16.5 fps |
| 44 | 59484.5 $\mu$s | 16.9 fps | 63430.3 $\mu$s | 15.8 fps |
| 45 | 61233.6 $\mu$s | 16.4 fps | 65608.7 $\mu$s | 15.2 fps |
| 46 | 60910.8 $\mu$s | 16.5 fps | 67441.7 $\mu$s | 14.9 fps |
| 47 | 64527.9 $\mu$s | 15.5 fps | 75150.3 $\mu$s | 13.3 fps |
| 48 | 66530.0 $\mu$s | 15.1 fps | 84304.2 $\mu$s | 11.9 fps |
| 49 | 65906.4 $\mu$s | 15.3 fps | 88711.2 $\mu$s | 11.3 fps |
| 50 | 66008.1 $\mu$s | 15.2 fps | 94406.7 $\mu$s | 10.6 fps |
| 51 | 69385.1 $\mu$s | 14.5 fps | 103708.7 $\mu$s | 9.7 fps |
| 52 | 72069.2 $\mu$s | 13.9 fps | 108344.3 $\mu$s | 9.2 fps |
| 53 | 71376.2 $\mu$s | 14.1 fps | 110035.7 $\mu$s | 9.1 fps |
| 54 | 72089.6 $\mu$s | 13.9 fps | 115906.0 $\mu$s | 8.6 fps |
| 55 | 71465.4 $\mu$s | 14.1 fps | 120949.7 $\mu$s | 8.3 fps |
| 56 | 70515.2 $\mu$s | 14.3 fps | 121203.0 $\mu$s | 8.3 fps |
| 57 | 72563.6 $\mu$s | 13.8 fps | 124720.7 $\mu$s | 8.0 fps |
| 58 | 75455.6 $\mu$s | 13.3 fps | 128808.0 $\mu$s | 7.8 fps |
| 59 | 78347.3 $\mu$s | 12.8 fps | 130304.3 $\mu$s | 7.7 fps |

## 11 References

## References

Andersson, J (2009). *Parallel Graphics in Frostbite-Current and Future.* [online]. URL: `http://s09.idav.ucdavis.edu/talks/04-JAndersson-ParallelFrostbite-Siggraph09.pdf` (visited on 08/27/2014).

Anon (1994). *Doom Review.* [online]. URL: `http://www.edge-online.com/review/doom-review/` (visited on 08/27/2014).

Deering M. Winner, S. Et Al (1988). *The Triangle Processor and Normal Vector Shader: a VLSI system for high performance graphics.* [online]. URL: `http://www.cs.unc.edu/techreports/92-034.pdf` (visited on 08/27/2014).

Engel, W. (2013). *GPU Pro 4.* CRC Press.

Filion D. McNaughton, R. (2008a). *Starcraft 2 Effects and Techniques.* [online]. URL: `http://developer.amd.com/wordpress/media/2013/01/Chapter05-Filion-StarCraftII.pdf` (visited on 08/27/2014).

Hargreaves, S (2004a). *Deferred Shading, EVOLVE Games Developers Conference.* [online]. URL: `http://www.shawnhargreaves.com/DeferredShading.pdf` (visited on 08/27/2014).

Harris M. Hargreaves, S. (2004). *6800 Leaguges Under The Sea.* [online]. URL: `http://tinyurl.com/ogrrkpv` (visited on 08/27/2014).

Lee, M (2009). *Pre-lighting in Resitance 2.* [online]. URL: `http://www.insomniacgames.com/tech/articles/0409/files/GDC09%5C_Lee%5C_Prelighting.pdf` (visited on 08/27/2014).

Phong, T (1973). *Illumination for Computer-Generated Images.* [online]. URL: `http://www.dtic.mil/dtic/tr/fulltext/u2/a008786.pdf` (visited on 08/27/2014).

Shishkovtsov, O. (2005). *Deferred Shading in S.T.A.L.K.E.R. Addison Wesley.* [online]. URL: `http://http.developer.nvidia.com/GPUGems2/gpugems2%5C_chapter09.html` (visited on 08/27/2014).

Tebbs, B. Et Al (1992). *Parallel Architectures and Algorithms for Real-Time Synthesis of High Quality Image using Deferred Shading.* [online]. URL: `http://www.cs.unc.edu/techreports/92-034.pdf` (visited on 08/27/2014).

### 11.1 Images

## References

Filion D. McNaughton, R. (2008b). *Starcraft 2 Effects and Techniques.* [online]. URL: `http://developer.amd.com/wordpress/media/2013/01/Chapter05-Filion-StarCraftII.pdf` (visited on 08/27/2014).

Hargreaves, S (2004b). *Deferred Shading, EVOLVE Games Developers Conference.* [online]. URL: `http://www.shawnhargreaves.com/DeferredShading.pdf` (visited on 08/27/2014).

Meiri, E (2014). *Deferred Shading - Part 3*. [online]. URL: http://ogldev.atspace.co.uk/www/tutorial37/tutorial37.html (visited on 08/27/2014).

## 12 Bibliography

## Bibliography

Andersson, J (2009). *Parallel Graphics in Frostbite-Current and Future.* [online]. URL: http://s09.idav.ucdavis.edu/talks/04-JAndersson-ParallelFrostbite-Siggraph09.pdf (visited on 08/27/2014).

Anon (1994). *Doom Review.* [online]. URL: http://www.edge-online.com/review/doom-review/ (visited on 08/27/2014).

Corporation, Microsoft (2004). *Direct X11 SDK[computer program].* [online]. URL: http://www.microsoft.com/en-gb/download/details.aspx?id=6812 (visited on 08/27/2014).

Corporation, Microsoft (2013). *Visual Studio 2013 [computer program].* [online]. URL: http://msdn.microsoft.com/en-gb/vstudio/aa718325(v=vs.110).aspx (visited on 08/27/2014).

Dazzlewalla (2009). *T-Rex[Note].* [online]. URL: http://www.turbosquid.com/FullPreview/Index.cfm/ID/449755 (visited on 08/27/2014).

Deering M. Winner, S. Et Al (1988). *The Triangle Processor and Normal Vector Shader: a VLSI system for high performance graphics.* [online]. URL: http://www.cs.unc.edu/techreports/92-034.pdf (visited on 08/27/2014).

Engel, W. (2013). *GPU Pro 4*. CRC Press.

Filion D. McNaughton, R. (2008a). *Starcraft 2 Effects and Techniques.* [online]. URL: http://developer.amd.com/wordpress/media/2013/01/Chapter05-Filion-StarCraftII.pdf (visited on 08/27/2014).

Filion D. McNaughton, R. (2008b). *Starcraft 2 Effects and Techniques.* [online]. URL: http://developer.amd.com/wordpress/media/2013/01/Chapter05-Filion-StarCraftII.pdf (visited on 08/27/2014).

Hargreaves, S (2004a). *Deferred Shading, EVOLVE Games Developers Conference.* [online]. URL: http://www.shawnhargreaves.com/DeferredShading.pdf (visited on 08/27/2014).

Hargreaves, S (2004b). *Deferred Shading, EVOLVE Games Developers Conference.* [online]. URL: http://www.shawnhargreaves.com/DeferredShading.pdf (visited on 08/27/2014).

Harris M. Hargreaves, S. (2004). *6800 Leaguges Under The Sea.* [online]. URL: http://tinyurl.com/ogrrkpv (visited on 08/27/2014).

Intel (2014). *Intel Graphics Performance Analyzers [computer program].* [online]. URL: https://software.intel.com/en-us/vcsource/tools/intel-gpa (visited on 08/27/2014).

Lee, M (2009). *Pre-lighting in Resitance 2.* [online]. URL: http://www.insomniacgames.com/tech/articles/0409/files/GDC09%5C_Lee%5C_Prelighting.pdf (visited on 08/27/2014).

Luna F, D (2012). *Introduction to 3D Game Programming with Direct X11*. Mercury Learning and Information.

Meiri, E (2014). *Deferred Shading - Part 3*. [online]. URL: `http://ogldev.atspace.co.uk/www/tutorial37/tutorial37.html` (visited on 08/27/2014).

Phong, T (1973). *Illumination for Computer-Generated Images*. [online]. URL: `http://www.dtic.mil/dtic/tr/fulltext/u2/a008786.pdf` (visited on 08/27/2014).

Shishkovtsov, O. (2005). *Deferred Shading in S.T.A.L.K.E.R. Addison Wesley*. [online]. URL: `http://http.developer.nvidia.com/GPUGems2/gpugems2%5C_chapter09.html` (visited on 08/27/2014).

Tebbs, B. Et Al (1992). *Parallel Architectures and Algorithms for Real-Time Synthesis of High Quality Image using Deferred Shading*. [online]. URL: `http://www.cs.unc.edu/techreports/92-034.pdf` (visited on 08/27/2014).