# An Evaluation of Deferred Shading Under Changing Conditions

Bart Postma

University of Groningen

## 1 Introduction

Deferred shading is a relatively new shading technique and it has gained increasing attention the last few years. Yet, the information about its performance is often incomplete or regards a specific application. The research presented in this paper wants to fill a number of knowledge gaps regarding this problem.

An often quoted reference in literature about deferred shading is the presentation of Hargreaves and Harris [5]. This presentation illustrates the basics of deferred shading, but not more than that. In [9] and [6] it is explained how deferred shading has been implemented in two recent computer games. Computer games, however, have many application-specific optimizations, not suited for all applications. The information in [9] and [6], however, is valuable for possible optimizations and pitfalls of deferred shading. An elaborate paper about deferred shading is [4]. It explains the principles of deferred shading, presents an implementation in OpenGL, a few very basic performance tests, and how to handle several effects such as shadows, anti-aliasing, etc.

This paper wants to extend the research done in the above mentioned sources, by filling a number of knowledge gaps that exists about deferred shading. Deferred shading can give a significant performance increase, however it can also give a significant performance decrease. The performance is highly dependable on a number of conditions. This paper presents how well deferred shading performs when these conditions change. The tested conditions concern the number of lights, the type of lights, the illumination model, the complexity of the scene, and the influence range of lights. Furthermore, the obtained results are compared with the results of traditional shading.

This paper will start with an explanation of deferred shading. This is followed with an implementation of traditional and deferred shading, presented at a high-abstraction level. The implementation has efficiency as a high priority, but does not perform application-specific optimizations. The next section will explain the experiment setup and is followed by the results. We end with a discussion of the results, future work, and conclusions.

### 1.1 Terminology

Throughout this paper, a number of terms are used. For a good understanding of this paper, a brief explanation of them is presented here first.

A 3D object is constructed with surface primitives. Surface primitives are specified with vertices, where each vertex has several geometry attributes, e.g. position, color, normal, material properties, etc. A scene is a collection of 3D objects, or put otherwise, a scene is a

collection of positions, colors, normals, material properties, etc.

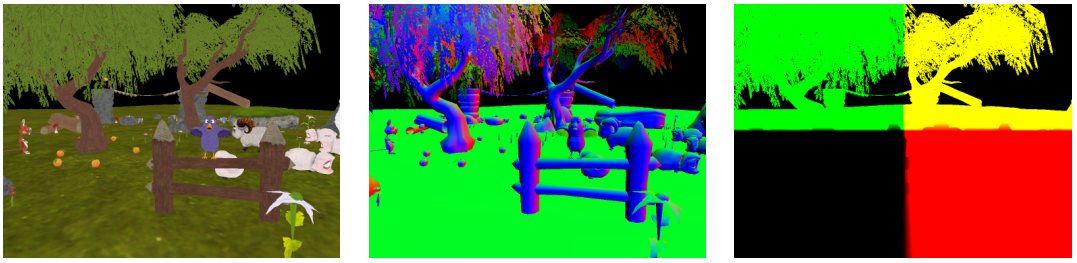
# 2 What is deferred shading

Synonyms for 'deferred' are 'delayed' or 'postponed'. The name 'deferred shading' hints to something being done later, and this is the key concept of deferred shading. With traditional shading, the scene is rendered and directly shaded in one rendering pass. With deferred shading, a first rendering pass renders the scene to a buffer without performing shading computations. More concretely, attributes as position, color, normal, material properties, etc. are rendered to a buffer. When this is done, the buffer contain the scene's projection observed from the current view point. In a second rendering pass, the shading computations are performed as a 2D post-process, by using the geometry attributes stored in the buffer from the first rendering pass. This has the advantage that shading computations are only performed on parts of the scene that are visible in the final image.

Deferred shading shades the absolute minimum of the scene, whereas traditional shading might shade parts of the scene which are not visible in the final image. The performance penalty from deferred shading comes from the requirement of rendering the scene to a buffer first and later retrieving the geometry attributes again from this buffer. It requires a very high bandwidth between the GPU and video memory.

## 2.1 G-buffer

The first rendering pass of deferred shading renders the geometry attributes of the scene to a buffer. The buffer in which these attributes are stored, is usually called the G-buffer [8]. Most illumination models require the following geometry attributes: position, color, normal, and material properties. Therefore, these four geometry attributes have to be rendered to the G-buffer. Figure 1 shows an example of the color, position, and normal buffer of a scene.



(a) Color buffer. (b) Normal buffer (in eye space). (c) Position buffer (in eye space).

Figure 1: Color, position, and normal buffer of a scene.

## 2.2 Shading

The G-buffer contains the scene's projection observed from the current view point. The second rendering pass performs the shading. The approach for doing this is as follows:

1. Determine the light's influence range on the scene's projection.
2. Retrieve the corresponding attributes from the G-buffer and perform shading.

The first step requires the projection of the light's influence range. The second step is practically identical to the shading computations of traditional shading. For performance reasons, the influence range of a light is usually restricted to a certain volume, the light volume. This is under the assumption that the light's influence outside this volume is negligible. To obtain the result of step 1, we first have to construct the light volume of the light source. This light volume is then processed, but not displayed, instead its projection is used for step 2. Or more concretely, the generated fragments of the light volume's projection are used for step 2 to retrieve the corresponding attributes from the G-buffer.

If we first perfrom ambient shading with an initial rendering pass. Then we can obtain the added shading contributions of all light sources by performing the above two steps with additive blending enabled.

## 2.3 Light volumes

The fact that we need the light volume to perform shading leads us to the point of determining how to represent the light volume for various types of lights. Usually, three types of lights are distinguished: point lights, spot lights, and directional lights, each having a characteristic light volume.

The influence range of a point light is spherical, it is assumed that it only influences objects that are within the volume of the sphere. A strong emitting point light will have a larger sphere than a weak emitting point light. With deferred shading, this sphere is projected onto the screen, but instead of displaying the sphere, we use the generated fragments to retrieve the corresponding attributes from the G-buffer and compute the shading. That is, we compute the distance from the light's position to the retrieved position from the G-buffer. If this distance is smaller than the light's influence radius we compute the shading, otherwise we discard the fragment. Figure 2 illustrates the process.

(a) Ambient shaded. (b) Projection of three spheres. (c) Fully shaded.

Figure 2: Using the spheres' projections to compute the shading.

The second type of light source, is the spot light. The influence range of a spot light is assumed to be a cone, it only influences objects that are within the volume of the cone. A strong emitting spot light will extend further than a weak emitting spot light. Note that a stronger emitting spot light will not make the spot beam wider, it is the height of the cone that becomes larger. Figure 3 illustrates the process of using the cones' projections to compute the shading with spot lights.

A directional light does not have a location, only a direction. A directional light is assumed to be very far away and therefore only its direction is relevant. The influence range of a directional light affects the whole scene, therefore it is also unnecessary to perform checks whether an object is within its influence range, this is always the case. Its influence on the scene's projection can be represented with a full-screen quad, because of its influences on

(a) Ambient shaded. (b) Projection of three cones. (c) Fully shaded.

Figure 3: Using the cones' projections to compute the shading.

the whole scene. Figure 4 illustrates the process of using full-screen quads to compute the shading with directional lights.

(a) Ambient shaded. (b) Full-screen quad. (c) Fully shaded.

Figure 4: Using a full-screen quad to compute the shading.

Besides point, spot and directional lights, other light sources are also possible. If we, for example, want a light source consisting of a line, then its influence range can be considered a bounding cylinder around this line. In principle, any type of light source is possible. The type of light source does not change the shading computations, only the influence range of the light source changes.

# 3 Implementation details

We proceed to an OpenGL implementation of both traditional and deferred shading, and why certain decisions have been made regarding the implementation. For a fair comparison between traditional and deferred shading we cannot have the situation where the CPU is the bottleneck. Therefore, both shading techniques will run solely on the GPU. The CPU is only used for setting things up, all the actual computations are performed on the GPU.

The scene and the light sources are stored in Vertex Buffer Objects (VBOs). To prevent caching, an animation is performed with the light sources. This is done by passing the light sources to a vertex program which performs an animation on the light positions and then streams the results back to a second VBO (in the next pass, the second VBO is used as input and the first one for the result, etc.).

## 3.1 Traditional shading

Figure 5 shows a schematic illustration of how traditional shading has been implemented.

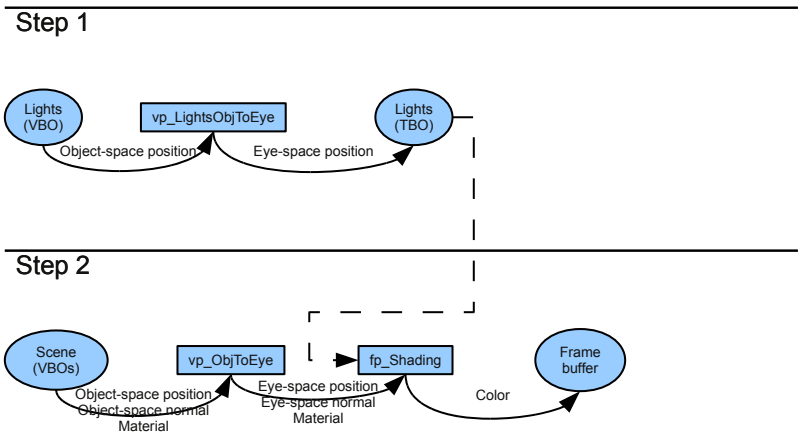



Figure 5: Implementation of traditional shading.

Shading is performed in eye space. Therefore, the first step is transforming the animated light positions to eye space with a vertex program. The results are stored in a Texture Buffer Object (TBO) for later use by the fragment program performing the shading.

The second step processes the scene. The scene is passed to the vertex program `vp_ObjToEye`, which transforms the vertex position with the model-view-projection matrix and also outputs the eye-space position, eye-space normal, color, and material properties to fragment program `fp_Shading`. The fragment program loops over all light positions in the TBO, computes the shading contribution of each light and adds this to the output color (see Listing 1 for pseudo-code of `fp_Shading`). In the case of spot lights, there will be a second TBO with spot directions.

```
outColor = (0.0, 0.0, 0.0)
for (i = 0; i < numLights; ++i) do
  lightPos = texBUF(tboLightPos, i)
  if (fragment within influence of light) then
    shading = computeShading()
    outColor += shading
  end
end
```

Listing 1: Traditional shading.

## 3.2 Deferred shading

Figure 6 shows a schematic illustration of how deferred shading has been implemented.

Before we start, we create a Frame Buffer Object (FBO) and attach screen-sized textures to it for each necessary geometry attribute (the G-buffer). Deferred shading consists of two rendering passes, the first rendering pass must render the scene to the G-buffer. To do so, we bind the FBO, everything is now rendered to its attached textures. Then the scene is passed to vertex program `vp_ObjToEye`, which transforms the position with the model-view-projection matrix and also outputs the eye-space position, eye-space normal, color, and material properties to fragment program `fp_Gbuffer`. Fragment program `fp_Gbuffer`

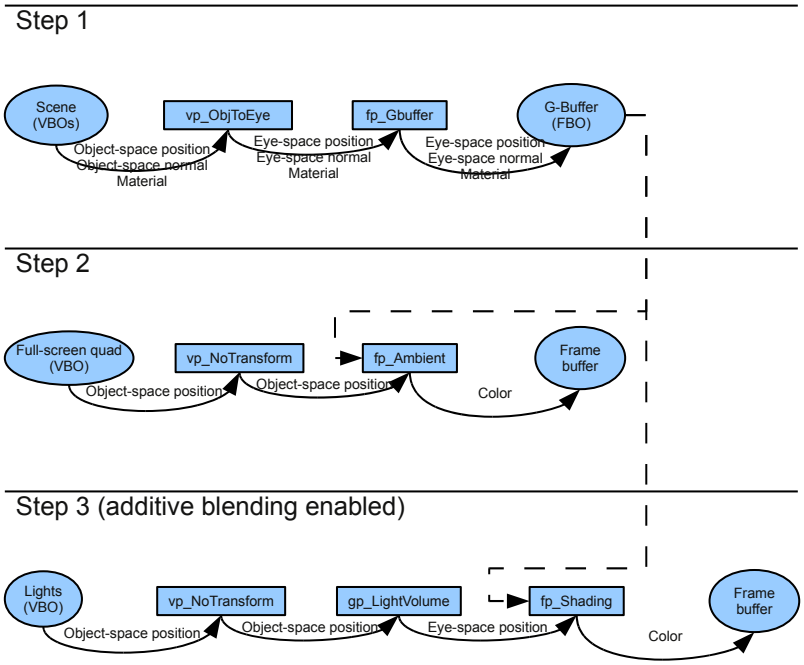



Figure 6: Implementation of deferred shading.

outputs the geometry attributes in such a way that they are rendered to the correct texture of the FBO. The G-buffer's textures now contain the scene's projection observed from the current view point.

After the construction of the G-buffer, the shading is performed. Ambient shading influences the entire scene and does not depend on a light source. Fragment program `fp_Ambient` receives a full-screen quad, denoting an influence on the entire projection of the scene. The fragment program retrieves the corresponding color from the G-buffer and multiplies the color with the ambient factor. The resulting ambient shaded scene is stored in the frame buffer.

Next we compute the shading contributions of the lights, and add the contributions to the ambient shaded scene by enabling additive blending. We pass the VBO containing the light sources to the vertex program `vp_NoTransform`, which does not perform any computation, it only outputs its data to the geometry program `gp_LightVolume`. Depending on the type of light source, this program constructs the corresponding light volume from the light's position (and spot direction in the case of spot lights). That is, it constructs a sphere for point lights, a cone for spot lights, and a full-screen quad for directional lights. `gp_LightVolume` applies the model-view-projection matrix to the constructed light volume, resulting in it being projected onto the screen and then rasterized into fragments. These fragments are received by the fragment program `fp_Shading`, which uses them to retrieve the corresponding geometry attributes from the G-buffer's textures and compute the shading.

### 3.2.1 Organizing the G-buffer

A texel consists of four components, i.e. red, green, blue, and alpha. From a performance perspective, it is better to minimize the number of G-buffer textures. Instead of having a separate texture for each geometry attribute, we will organize the components of the G-buffer

textures more efficiently. In [9] a number of organizations are presented, together with their advantages and disadvantages. For example, by encoding certain attribute components, we can save memory. We are not limited in memory, therefore we will use an organization that does not require encoding and decoding. The used G-buffer organization is presented in Table 1. All components are stored as 16 bit floating-point numbers, as is found in the majority of applications using deferred shading.

| Component | Texture 1 | Texture 2 | Texture 3 |
|---|---|---|---|
| Red | Diffuse color Red | Position X | Normal X |
| Green | Diffuse color Green | Position Y | Normal Y |
| Blue | Diffuse color Blue | Position Z | Normal Z |
| Alpha | Ambient factor | Specular factor | Shininess |

Table 1: Organization of the G-buffer.

### 3.2.2 Optimizing the geometry program

Constructing complex objects, such as spheres and cones, in a geometry program comes at a cost, therefore it is worthwhile to optimize this step. The geometry program `gp_LightVolume` outputs a simpler light volume for a spot light and point light than a cone and sphere respectively.

For a spot light, not a cone is constructed, but a four-sided pyramid that bounds the cone (see Fig. 7). This requires the construction of only four triangles. The base of the pyramid does not need to be constructed, since it is only the pyramid's projection we are interested in.

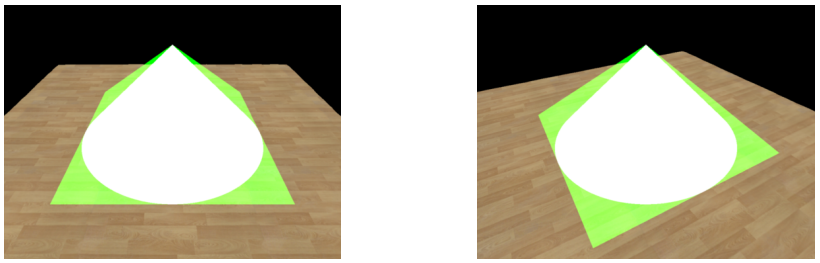


Figure 7: A green four-sided pyramid bounding a white cone.

For a point light, a bounding cube can be constructed around the sphere, this requires the construction of twelve triangles. However, we can obtain the same end result with only two triangles. This is possible because only the projection of the light volume is relevant. A billboard (two triangles) is constructed that bounds the sphere's projection. A billboard is always directed towards the viewpoint, so it will bound the sphere's projection from all view points (see Fig. 8).

# 4 Experiment setup

In the experiment we want to compare the performance of deferred shading with traditional shading and observe how they scale under changing conditions. These conditions concern

Figure 8: A green billboard bounding a white projected sphere.

the number of lights, the complexity of the scene, and the influence range of lights. The experiment is performed with four well-known illumination models: Phong [7], Blinn-Phong [1], Cook-Torrance [2], and Gooch illumination [3]. Secondly, three types of light sources are used: spot lights, point lights with local influence, and point light with global influence. Point lights with global influence affect the whole scene. For the sake of the animation on the lights, we use point lights with global influence instead of directional lights. From a performance perspective, this change is negligible. The change is that the light direction is now determined from the object's location and the lights's location and distance attenutation is performed.

The scene being used for the experiment tries to reflect the complexity level of scenes found in graphics applications nowadays. It consists of $1.2 \cdot 10^5$ vertices and uses a total of 23 textures, roughly half of the textures are of size $512 \times 512$ and the other half of size $1024 \times 1024$. Figure 9 shows the scene from two view points.

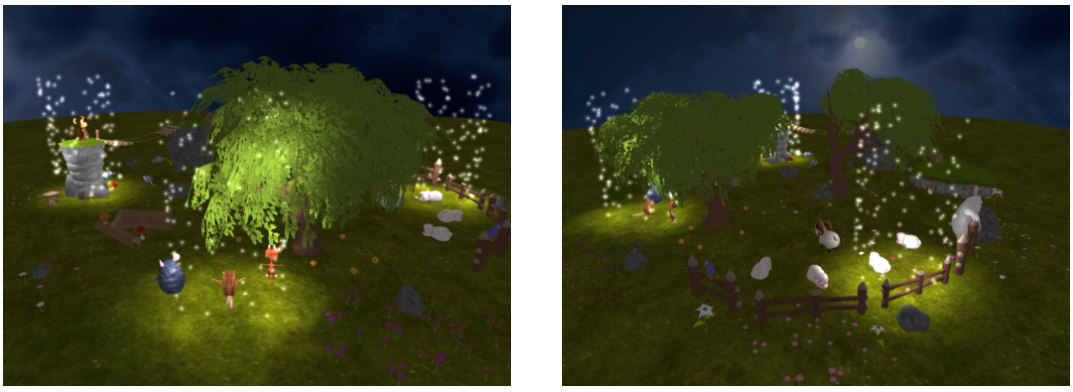


Figure 9: Test scene viewed from two different view points.

A good measure for the performance of real-time applications is the frame rate. The tests will vary one condition, while keeping the other conditions fixed, and measure the frame rate. The results are presented with a graph for each illumination model and each type of light. A discussion regarding the results will follow after all results have been presented.

The experiment requires us to make a number of assumptions about the properties of the lights. The assumptions try be a reasonable reflection of what can be found in today's graphics applications. This is subjective to personal experience and view to what is reasonable. A few tests require a number of additional assumptions, which are mentioned when they are applied. We use the knowledge that almost all objects in the scene have an $x, z$-position within a radius of 100 from the scene's center (in object space). Secondly, the objects do not

exceed a height of $y = 50$. With this knowledge, the following assumptions are made about the light properties:

- The $x, z$-positions of the lights are located within a radius of 100 from the scene's center and their height is uniformly distributed between 3 and 50.
- Point lights with local influence have an influence radius of size 20.
- Spot lights are directed downward and have a cut off angle of 45°. The cut off angle is a value between 0° and 90° and denotes how narrow the beam of a spot light is, where a lower value means more narrow.

The tests are performed with a resolution of $1280 \times 1024$. The system being used is an Intel Core 2 Duo E6550 2.33 GHz, 2 GB DDR2 RAM, and an Nvidia Geforce 8600 GT.

# 5 Results

## 5.1 Number of lights

In the first test, the number of lights are varied. This is done by starting with no lights and increasing it to 50 lights. 50 lights will prove to be high enough to see how traditional and deferred shading scale with the number of lights. The results are shown in Figure 10.

## 5.2 Scene complexity

The second test concerns the scene complexity, with respect to the number of generated fragments. Early testing of the application revealed that its performance is bound by the number of fragments being processed. Therefore, we choose to measure the scene complexity in number of generated fragments instead of number of vertices. The test will start with one 3D object in the scene and additional 3D objects are added to the scene until the scene reaches full complexity as depicted in Figure 9.

With the `ARB_occlusion_query` extension in OpenGL, it is possible to determine the number fragments generated by the scene. The extension can be used to obtain the number of generated fragments of a scene after the depth test. By disabling depth testing, we obtain the total number of generated fragments of a scene processed by a fragment program.

This test requires us to make an assumption about the number of lights to use. Based on personal experience and what seems reasonable, we test with 15 spot lights, 15 point lights with local influence, and 3 point lights with global influence. The results are presented in Figure 11. Note that this test is more appropriate for observing the scaling of the two shading techniques and not for comparing the frame rates with each other. This is because the first test (comparing frame rate with number of lights) already revealed that, for example, deferred shading is faster with 15 spot lights than traditional shading.

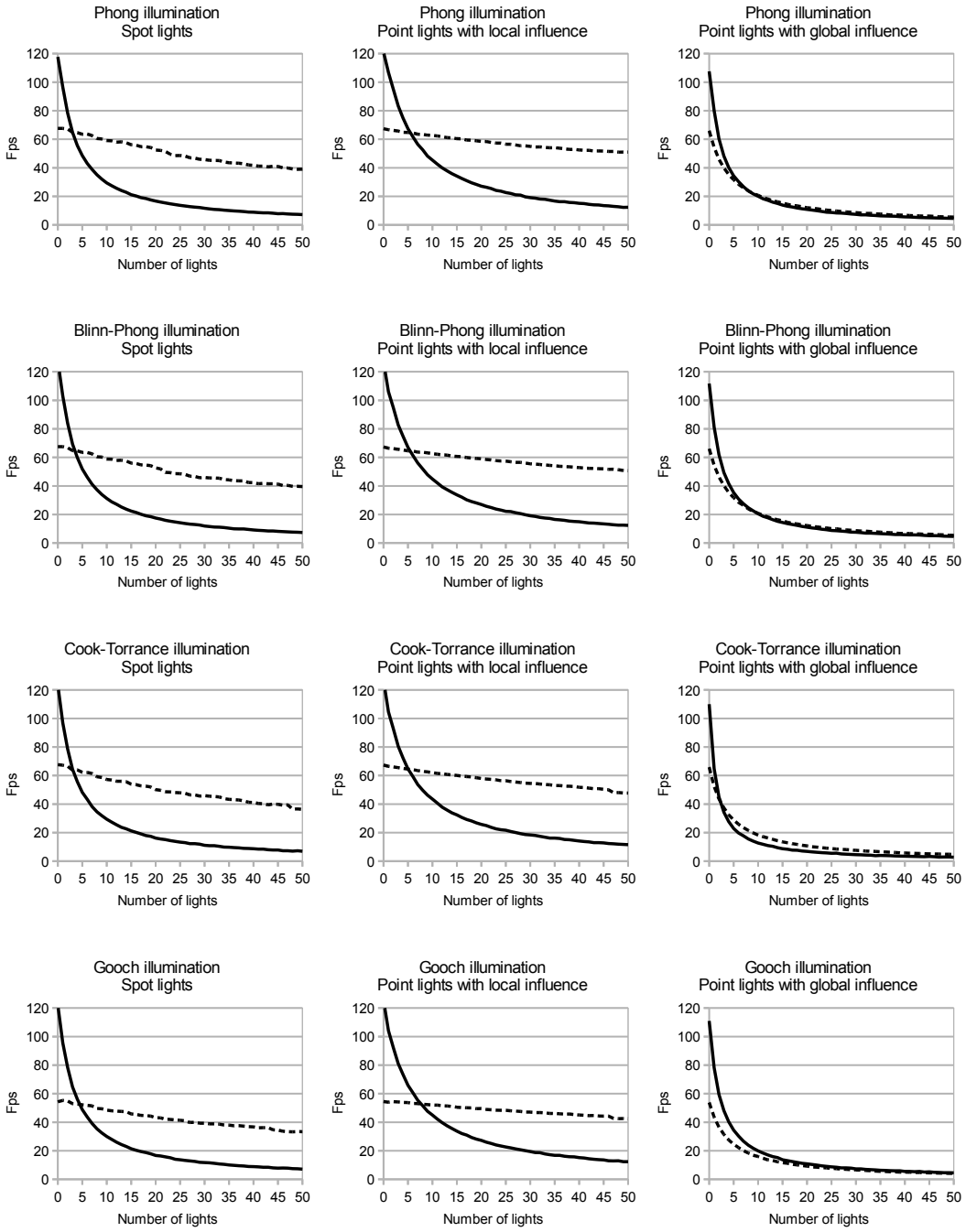



Figure 10: Frame rate versus the number of lights. The dashed line corresponds with deferred shading and the continuous line corresponds with traditional shading.

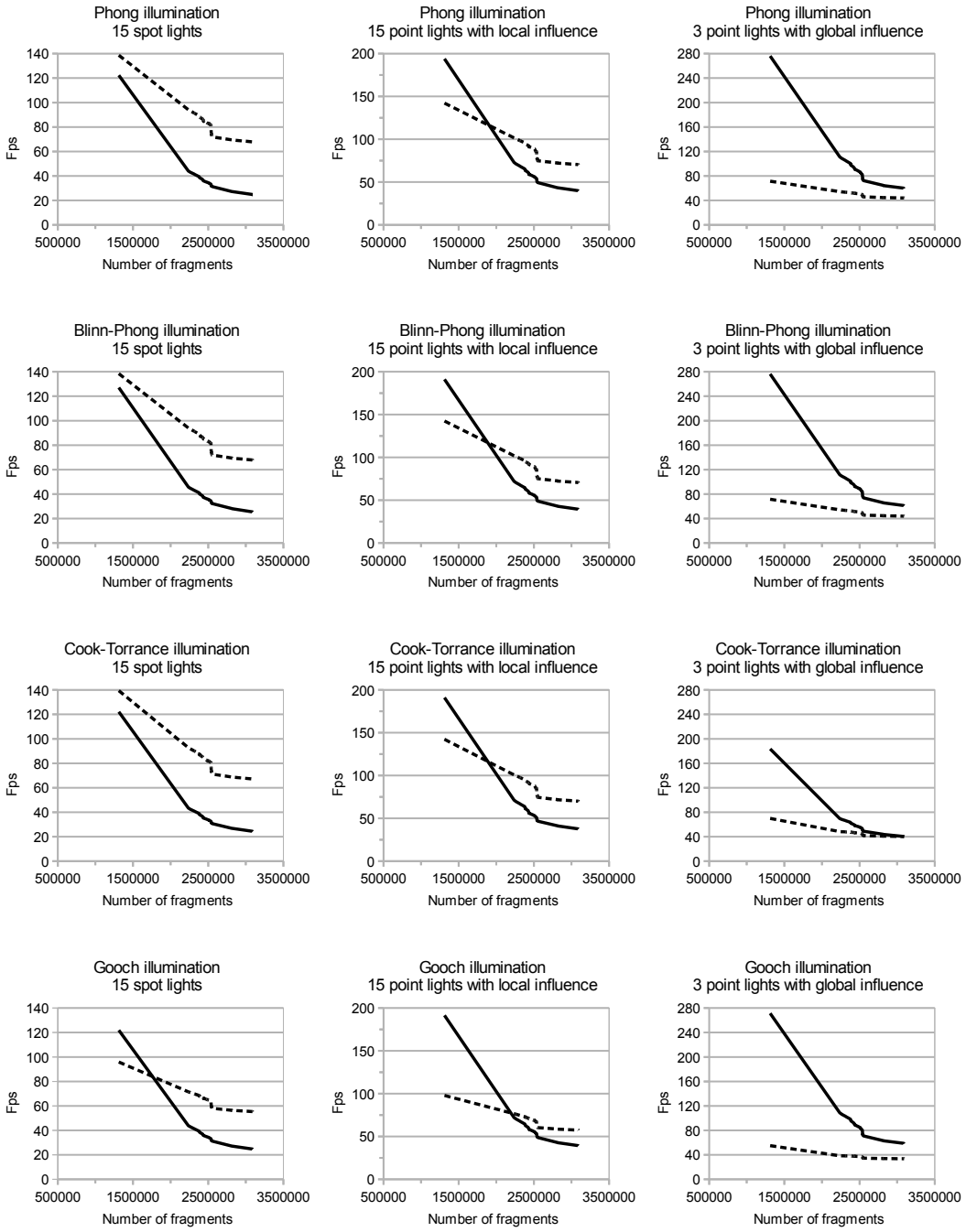



Figure 11: Frame rate versus the number of lights. The dashed line corresponds with deferred shading and the continuous line corresponds with traditional shading.

## 5.3 Influence range of light source

Observing the graphs from the previous tests, we see a significant difference between lights with local influence (spot lights and point lights with local influence) and lights with global influence. The final test will measure the frame rate versus the influence range of a light source. We start with a number of point lights with a small influence radius and gradually increase their influence radius.

Initially, the point lights have a relatively small influence radius of size 10 (in object space). We increase the influence radius in ten steps in such a way that the light's final influence radius affects almost all the objects in the scene. An influence radius of size 110 affects almost all objects in the scene, therefore we will add 11 to the influence radius each step.

We make the assumption that five lights is a reasonable number of lights. The results are shown in Figure 12. The second test (comparing frame rate with scene complexity) was more appropriate for observing the scaling of the two shading techniques and not for comparing the frame rates with each other. The same reason for doing that also holds for this test.

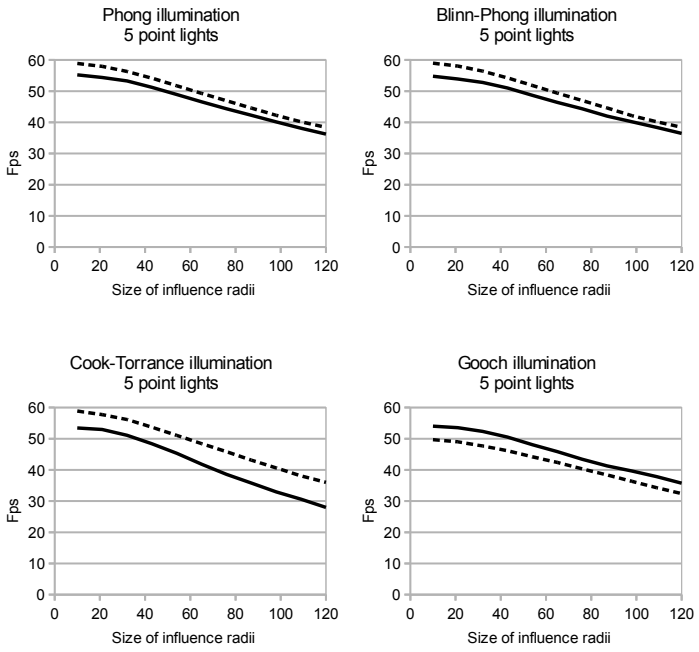



Figure 12: Frame rate versus the influence radii of five point lights. The dashed line corresponds with deferred shading and the continuous line corresponds with traditional shading.

# 6 Discussion

The test in which the two shading techniques differ most, is the test which changes the number of lights and the lights having local influence, i.e. spot lights and point lights with local influence. The most remarkable observation here, is the relative insensitivity of deferred shading to the number of lights compared to traditional shading. The scaling of traditional

shading shows an exponential decay with all illumination models and types of light. If we take a closer look to Phong illumination and point lights with local influence, then deferred shading shows a more or less linear decay. With the first ten lights, traditional shading's frame rate drops with 6.5 fps for each light, against 0.35 fps for each light with deferred shading. From 20 to 50 lights, both shading techniques show roughly a similar decay in frame rate for each added light. However, by then traditional shading has reached a relatively low frame rate. The reason for this scaling is that the G-buffer prevents that shading is performed on occluded objects, and more lights means a higher probability of spending shading computations on occluded objects. Deferred shading starts with a lower frame rate than traditional shading, which comes from the cost of constructing the G-buffer. With less than six lights, the cost of constructing the G-buffer does not outweigh the benefits it can give, using Phong illumination and point lights with local influence. With spot lights and point lights with global influence, this is three lights and seven lights respectively. Deferred shading does not have a noticeable advantage over traditional shading with lights having a global influence. Both shading techniques show an exponential decay, with deferred shading starting with a lower frame rate. The global influence causes many accesses of the G-buffer, which seem to limit deferred shading significantly.

The tests which measure frame rate with respect to the number of fragments have two things in common, i.e. they all show a linear decay and they all have a sudden drop in frame rate around $2.5 \cdot 10^6$ fragments. The linear decay is a consequence of the application being bound by the number of fragments. So adding more fragments to the scene, lowers the performance proportionally. The drop around $2.5 \cdot 10^6$ fragments can be explained by looking more closely to when and how many objects are drawn. Around that point, a number of objects consisting of many vertices are drawn. Although the application is bound by the number of fragments, it seems that the number of vertices is not entirely negligible.

The tests which relate frame rate with influence radius, do not reveal unexpected results. There is a linear decay of frame rate with the influence radius. A larger influence radius affects more objects and therefore more fragments, this lowers the frame rate proportionally.

Finally, there is one remarkable observation seen in all tests. That is, the relative low frame rate when performing deferred shading with the Gooch illumination model, whereas the illumination model is not significantly more expensive than Phong. The formula for Gooch illumination is as follows:

$$I = (1 - \mathit{diffuse}) \cdot \mathit{coolColor} + \mathit{diffuse} \cdot \mathit{warmColor} + \mathit{specular} \cdot \mathit{warmColor}.$$

The term $(1 - \mathit{diffuse}) \cdot \mathit{coolColor}$ might yield a color with a negative RGB-component. OpenGL clamps color components in the frame buffer between 0 and 1. This becomes a problem with deferred shading where the frame buffer receives the shading contributions of each light separately and alpha blending enabled. A negative shading contribution of one light is clamped to zero, whereas a second light with a positive contribution can be added to that. This yields a scene which is too bright. The solution is by first rendering to an intermediate floating-point FBO and then copy the end result from the FBO to the frame buffer. This solution requires an extra rendering pass, which explains the performance penalty. The test system runs Linux and testing on a Windows computer with an Nvidia Geforce 8800 GTX showed no noticeable performance penalty. It seems that graphics divers and/or graphics card are a factor of influence.

# 7 Future work

In [9] a property regarding the scene is mentioned, which can also be of influence in the choice between traditional and deferred shading. It is called *overdraw* and is defined as the number of fragments passing the depth test divided by the screen area. Deferred shading shades the absolute minimum of the scene, whereas traditional might shade parts that are not visible in the final image. Visible parts pass the depth test, therefore it can be interesting to see how both shading techniques perform and scale with scenes with an increasing amount of overdraw. It is possible to construct a scene consisting of full-screen quads, where each full-screen quad denotes an increase of 1 in the amount overdraw. However, performing this test revealed results that had a number of peculiarities that are inconsistent with the other tests and to what we would expect from both shading techniques. The scene consisting of full-screen quads only is probably not an appropriate scene for testing. Having more realistic scenes, each with a different amount of overdraw, would yield a better test.

What also would be interesting to know is the effect of texture compression. Texture compression lowers the bandwidth between GPU and video memory and deferred shading might benefit from this.

Thirdly, the current implementation uses a geometry program to construct a light volume. With a technique called *geometry instancing*, it is possible to draw multiple instances of a 3D object with a single draw call. Deferred shading has multiple instances of a light volume, so this technique might increase its performance.

# 8 Conclusions

From the results we can observe that both shading techniques have an ideal situation for their usage. Traditional shading is suited well for the situation when there are not much more than five lights and the scene complexity is relatively low. Deferred shading is a good choice for the situation when there are many lights, the lights have a local influence, and the scene is relatively complex.

The difficulty in choosing arises when the situation does not clearly belong to one of the two situations above or is unknown. The results show that deferred shading is less sensitive to the tested conditions than traditional shading. This makes deferred shading a safer choice when the situation is unknown or not entirely clear.

From a performance perspective, it can be a good idea to use both shading techniques. In scenes with lights having global influence and relatively few lights having local influence, we use traditional shading. In scenes with only a few lights having global influence, we use deferred shading. Combining the two shading techniques in the same scene may not be beneficial, because the scene then has to be processed twice, resulting in extra rendering passes.

The research presented in this paper focused on performance of the two shading techniques without performing application-specific optimizations. Depending on the application, several optimizations can be performed with both shading techniques. Some optimizations can give a significant improvement [9] [6]. Besides performance, other reasons can also be of influence in choosing between traditional or deferred shading. For example, ease of scene management, how to handle certain effects such as transparency, anti-aliasing, etc.

## References


[1] Blinn, J.F., *Models of Light Reflection for Computer Synthesized Pictures*, Proceedings of SIGGRAPH '77, p. 192-198, ACM Press, New York, USA, 1977

[2] Cook, R.L. and K.E. Torrance, *A Reflectance Model for Computer Graphics*, Proceedings of SIGGRAPH '81, p. 307-316, ACM Press, New York, USA, 1981

[3] Gooch, A, B. Gooch, P. Shirley, and E. Cohen, *A Non-Photorealistic Lighting Model For Automatic Technical Illustration*, Proceedings of SIGGRAPH '98, p. 447-452, ACM Press, New York, USA, 1998

[4] Gruber, L., *DeShade: A Deferred Shading Frame Work for Complex Lighting*, http://lux.surface.at/masterthesis/paper/Final_thesis_lukasgruber.pdf

[5] Hargreaves, S. and M. Harris, *Deferred Shading*, http://download.nvidia.com/developer/presentations/2004/6800_Leagues/6800_Leagues_Deferred_Shading.pdf

[6] Koonce, R., *Deferred Shading in Tabula Rasa*, GPU Gems 3, p. 429-457, Addison-Wesley Professional, 2007

[7] Phong, B.T., *Illumination for Computer Generated Pictures*, Communications of the ACM, Volume 18(6), p. 311-317, ACM Press, New York, USA, 1975

[8] Saito, T. and T. Takahashi, *Comprehensible Rendering of 3-D Shapes*, Proceedings of SIGGRAPH '90, p. 197-206, ACM Press, New York, USA, 1990

[9] Shishkovtsov, O., *Deferred Shading in S.T.A.L.K.E.R.*, GPU Gems 2, p. 143-165, Addison-Wesley Professional, 2005