UNIVERSITY OF LEEDS

School of Computing

FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

Final Report

A Comparative Study of Screen-Space Ambient Occlusion Methods

Sahil Puri

Submitted in accordance with the requirements for the degree of

BSc Computer Science

2023/24

COMP3931 Individual Project

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (16/08/224)*
Link to online code repository	*URL*	*Sent to supervisor and assessor (16/08/24)*

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) Sahil Puri _____

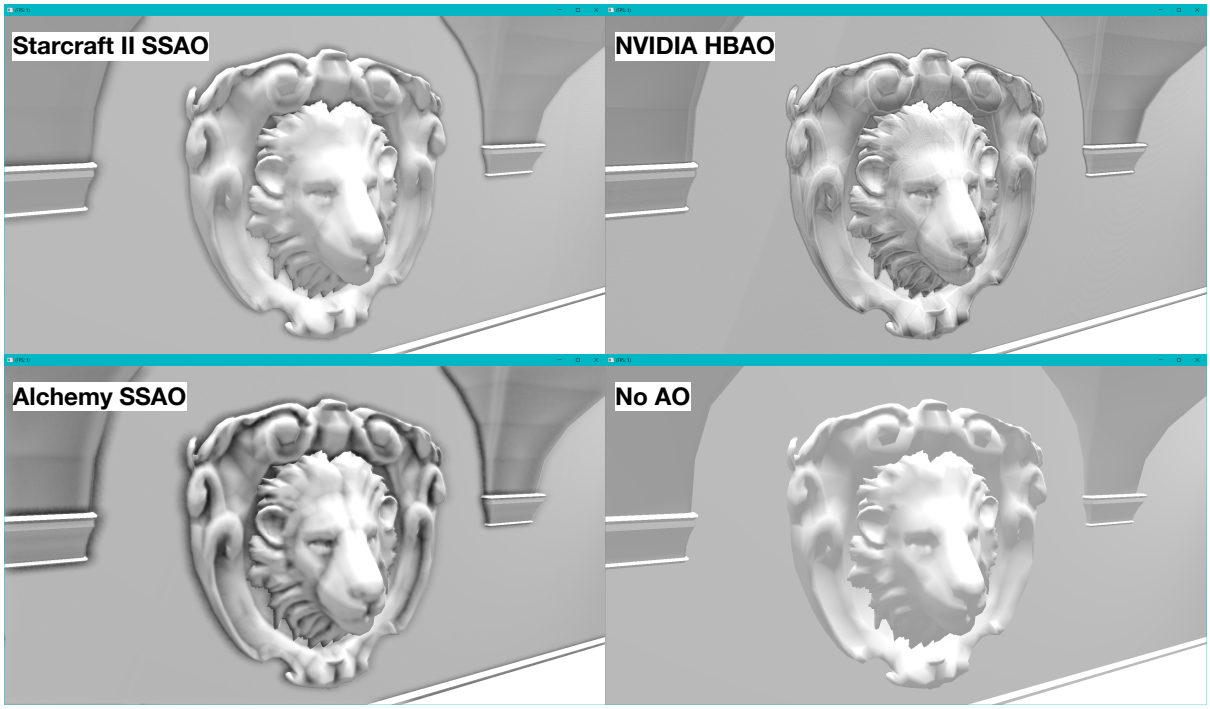
Summary

The goal of this dissertation is to evaluate and compare the performance and visual impact of different ambient occlusion techniques within a real-time rendering environment. Ambient occlusion is a rendering and shading method that determines the extent to which each point in a scene is exposed to ambient light. The significance of this study lies in its potential to enhance graphical fidelity in video games and simulations, contributing to more realistic and immersive virtual environments.

The research used a controlled virtual environment where a scene was set up to test different ambient occlusion settings. Four camera angles were established to capture various geometric and textural details within the scene. Three ambient occlusion techniques, SSAO, HBAO, Alchemy SSAO, and no occlusion to act as a normal, were applied and tested under each camera angle. Each technique was evaluated based on its performance over a five-second interval, repeated three times to ensure consistency and reliability. Performance metrics focused on frame rate measurements, and qualitative data was gathered through screenshot comparisons. Contrary to expectations, HBAO showed a higher frame rate than SSAO and Alchemy SSAO, suggesting a better optimization or more effective utilization of hardware resources in rendering scenes with HBAO. Alchemy SSAO provided superior visual enhancement, adding depth and realism especially noticeable in densely textured areas and complex spatial arrangements. However, Alchemy SSAO had the worst performance.

Although HBAO had the best performance, the visuals provided in this implementation failed compared to both other methods. The best method is dependent on the user's needs. Alchemy AO provides the best visual as well as easily adjustable parameters making it the best option for applications focused on realistic visuals. HBAO may be favored in applications that require better performance. However, the conclusions reached in this paper are limited by imperfect implementations of each technique and therefore should be considered in conjunction with other information.

Table of Contents



Chapter 1

Introduction

1.1 Introduction

Developers in computer graphics are constantly pursuing the goal of achieving realistically rendered scenes. This includes many endeavors such as, but not limited to, realistic lighting, textures, models, environments, and camera effects. One such method is using Ambient Occlusion (AO). It is a shading and rendering technique that calculates how exposed each point in a scene is to ambient lighting. It adjusts the brightness of points in a scene generating shadows and enhancing contours and details of objects resulting in a better sense of depth and visual clarity.

For real-time graphics applications, such as video games, the most popular ambient occlusion method is Screen Space Ambient Occlusion (SSAO) or variants of it, originally developed for the video game “Crysis” (Mittring, 2007). The SSAO method developed for Crysis inspired many new Screen Space methods which aimed to improve upon and optimize the original method like Starcraft II’s SSAO (Filion and McNaughton, 2008), NVIDIA’s Horizon Based Ambient Occlusion (Bavoli and Sainz, 2008) and Alchemy Screen Space Ambient Occlusion (McGuire et al., 2011). All of these are considered Screen Space methods because they compute ambient occlusion based on data available on the screenspace rather than the entire 3D scene. These methods will be explored in more depth in this paper.

1.2 Purpose

Screen Space Ambient Occlusion methods were adopted for video games due to their capabilities for real-time graphics applications. Compared to methods such as Ray Traced Ambient Occlusion or Precomputed/Baked Ambient Occlusion, SSAO methods can be used on applications with high

performance. It is generally faster and less computationally intensive than those other methods but less accurate. Regardless, it is considered ‘good enough’ for real-time graphics applications where performance is prioritized.

However, this doesn't mean that SSAO methods can't be further improved and optimized. Starcraft II’s SSAO, NVIDIA’s HBAO, and Alchemy SSAO all use different approaches to improving Screen Space Ambient Occlusion methods. All three methods aim for high performance and high accuracy using their unique approaches to occlusion. Each of these methods can be compared to assess the strengths and weaknesses of each method, thus helping create informed decisions over which method would best suit a developer’s graphics application.

1.3 Aims and Objectives

The primary aim of this project is to implement and compare Screen Space Ambient Occlusion methods in a graphics application. The three chosen methods to be compared are Starcraft II’s SSAO, HBAO, and Alchemy SSAO. These three methods are widely recognized in the industry and serve as benchmarks for evaluating the trade-offs between performance, visual fidelity, and computational complexity in Screen Space Ambient Occlusion. The details of how this comparison will be achieved will be explored in the method, but a brief outline of this project’s goals is shown below:

1. Set up a suitable graphics application to implement and compare the Ambient Occlusion methods.
2. Research implementations of all three methods to understand the algorithms.
3. Integrate all three methods into the application and fine-tune them for enhanced visual fidelity.
4. Evaluate the performance of all three methods:
    A. Measure the frames per second (FPS) of both methods under the same circumstances
    B. Compare the visual quality of all three methods using subjective visual analysis

Chapter 2
Background Research

This section will go over the research done for this project. Heavy focus will be put on four papers covering the Ambient Occlusion methods that will be explored in this project: Crytek’s SSAO (Mittring, 2007), Starcraft II’s SSAO (Filion and McNaughton, 2008), NVIDIA’s HBAO (Bavoli and Sainz, 2008) and Alchemy SSAO (McGuire et al., 2011). Each method will be looked at in depth to explore how each implementation works and can be implemented into a graphics application.

2.2 Crytek’s Screen Space Ambient Occlusion

Crytek was able to produce ambient occlusion through only screen space depth data. For every fragment on the screen, a sphere would be produced which is used for sampling. Then the method samples the depth buffers of points within the spheres. Then if the position of the sample is behind the sampled depth, it contributes to the occlusion factor. So the more points classified as inside geometry,

the more occluded the pixel and the darker it is (Mittring, 2007). Refer to Figure 2.1 for a better understanding of how it works.

In Figure 2.1 you can see 6 samples classified as visible and 10 samples classified as occluded. The occlusion factor can be calculated by finding what percentage of samples are occluded. Then you can change the brightness of the pixel using the occlusion factor.

$$\text{occlusion factor} = \frac{\text{occluded samples}}{\text{total samples}}$$

$$\text{birghtness} = 1 - \text{occlusion factor}$$

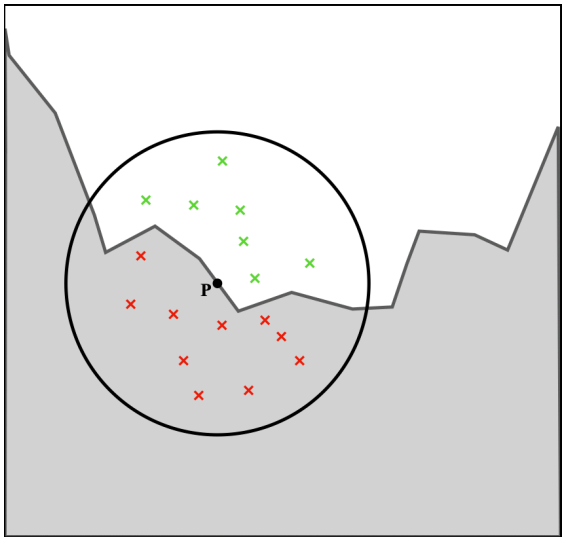


Figure 2.1 CryEngine 2 Screen Space Ambient Occlusion. Point p is the pixel that AO is being calculated for, the circle represents the sphere for sampling. The crosses represent the samples, green is visible and red is occluded.

Figure 2.2 SSAO Component of a typical game scene using Crytek's SSAO (Vlad3D, 2007)

This method offers an effective calculation for darkening corners and concave areas, darkening geometry in areas in a realistic manner. The method is also computationally cheap. It is a simple-to-understand method that is effective at computing occlusion for more realistic lighting. However, this method has a clear problem, on flat surfaces approximately half the samples will be occluded meaning every point has a higher occlusion factor. This issue is referred to as self-occlusion, resulting in darker scenes and a more uniform gray occlusion as opposed to sharper changes in brightness between dark and light spots. Figure 2.2 shows a scene rendered without textures. The shading on every fragment is a variation of gray, limiting the contrast the ambient occlusion would apply to the scene. Concave corners and obscured objects are still darkened, which is correct for realistic lighting but convex corners are brightened due to having less of the sampling sphere inside geometry, which is an unwanted effect. Ambient occlusion is a subtle effect and once textures are rendered these issues may not be as easily noticeable however there can always be improvements. There are many improved versions of Crytek's initial implementation of SSAO. One such improvement is the use of a normal-aligned hemisphere first implemented in *StarCraft II.*

2.3 StarCraft II's Screen Space Ambient Occlusion

This implementation of SSAO was done for the development of the video game *StarCraft II.* Filion and McNaughton revealed their implementation for SIGGRAPH 2008 with an in-depth explanation of how they overcame self-occlusion. The main adjustment to the implementation was the use of a hemisphere aligned with the normal of the point instead of a sphere for sampling. Otherwise, the method is the same as Crytek's SSAO. As you can see in Figure 2.3, the point is in the same spot as in Figure 2.1, but instead, we align the base of a hemisphere to be perpendicular to the point and sample that hemisphere. The point now has a much lower occlusion factor as fewer samples are occluded.

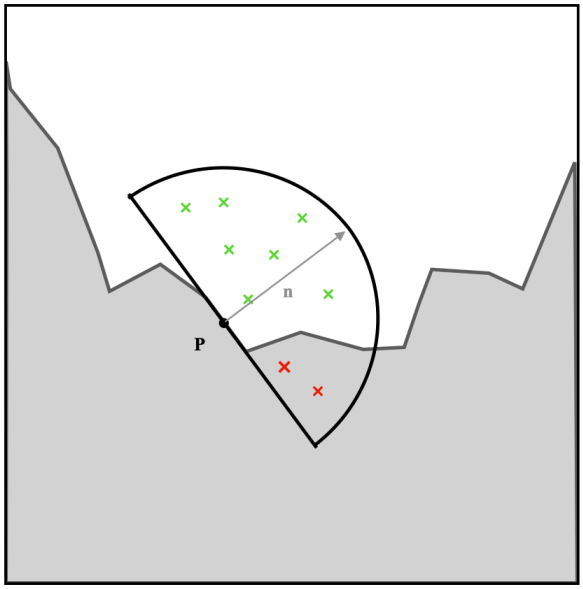


Figure 2.3 StarCraft II's Screen Space Ambient Occlusion. Point p is the pixel that AO is being calculated for, the semi-sphere represents the hemisphere for sampling. The crosses represent the samples, green is visible and red is occluded.

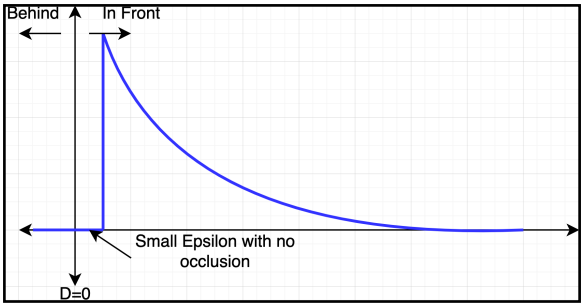


Figure 2.4 Starcraft II's SSAO occlusion function

Flat surfaces will have little to no occlusion and concave corners and surfaces partially blocked by other objects will still be darkened. This method removes the gray effect of the previous method and produces more realistic results in general. Thus this will be the SSAO method that will be implemented for this comparison as it is an almost direct improvement over Crytek's initial SSAO implementation.

The depth test however is not a simple boolean operation that determines if a sample point is in front of the the pixel. If a sample point is closer to the point in front of it, it should occlude the point more than once further away. Some points may be so far away that they shouldn't occlude the point at all. Thus this method implements an occlusion function mapping the relationship between the depth delta between the sampled point and its sampled depth and the amount of occlusion that occurs. The function is not set in stone and can be adjusted for the needs of the developers but Filion and McNaughton (2008) set criteria the function should adhere to:

" - Negative depth deltas should give zero occlusion (the occluding surface is behind the sample point);

- Smaller depth deltas should give higher occlusion values

- The occlusion value needs to fall to zero again beyond a certain depth delta value"
(Filion and McNaughton, 2008)

In their implementation, they opted for a linearly stepped function that can be controlled by the artist. Figure 2.4 visualizes the function. The main SSAO function and setup are fairly simple but there are also other details to improve the methods visuals.

Sampling randomization is an important feature that reduces artifacts, which are visual imperfections or unintended effects. One such approach discussed by Filion and McNaughton is to generate a 2D texture of random normal vectors and to use the texture in screen space giving a unique random vector for each pixel on the screen. This only gives one vector per pixel, so this randomly generated vector is used in tandem with a set of offset vectors to give each pixel on the screen randomized sampling. The next important step is to apply blur to further reduce noise.

For this implementation, Filion and McNaughton discussed the use of a 'smart' Gaussian blur that not only samples nearby pixels but their normals and depths too. Their Gaussian blur implementation

carries out an ordinary Gaussian blur but also has extra checks. If the depth of the Gaussian sample differs too much from the pixel, or the dot product of the Gaussian sample and the pixel's normal is less than a set value, the Gaussian weight is set to zero. Then the sum of the Gaussian samples is renormalized to account for the missing values from those samples.

Starcraft II's approach to ambient occlusion is heavily based on Mittring's original SSAO implementation but further improves it with many additional features and a better sampling method. The next two ambient occlusion methods that will be looked at deviate more from these first two screen space methods.

2.4 NVIDIA's Horizon-Based Ambient Occlusion

Figure 2.5 Example scene with a point highlighted by a star

Figure 2.6 Example scene with a point being highlighted by a star and a yellow rectangle highlighting the geometry shown in the height field

NVIDIA's Horizon-Based Ambient Occlusion (HBAO) is a sophisticated development in ambient occlusion techniques, which was designed to enhance the realism and depth perception in 3D graphics, particularly in video games. Similar to the adjustments made by Filion and McNaughton for StarCraft II, NVIDIA improved upon the standard SSAO by incorporating a horizon-based technique that better captures how ambient light interacts with objects in a scene.

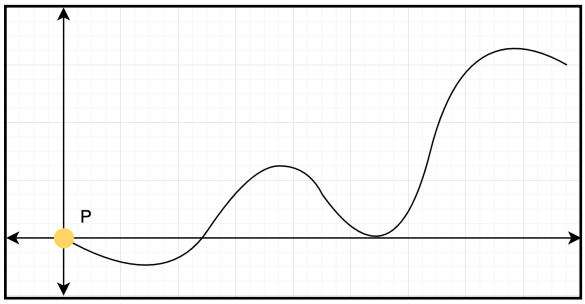


Figure 2.7 A 1-dimensional height field of the highlighted area in Figure 2.6. The yellow point P represents the highlighted star. The y-axis is the depth value of the geometry and the x-axis is the position of the geometry in the highlighted rectangle.

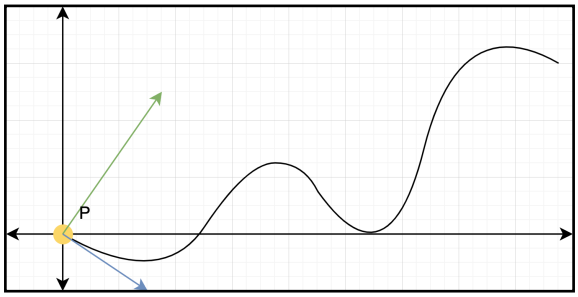


Figure 2.8 1-dimensional height field with the normal vector(green) and tangent vector(blue)

HBAO improves upon traditional SSAO methods by focusing on the way ambient light is occluded by nearby geometry, particularly taking into account the "horizon" angle created by objects relative to the point being shaded (Bavoli and Sainz, 2008). This method calculates the occlusion by considering the visible horizon line for each pixel, which allows it to more accurately and realistically simulate shadows in areas where light is partially blocked by surrounding objects.

Let us imagine we have a point we want to calculate the occlusion for. This point is highlighted by a star in Figure 2.5.

This point can be visualized in a 1-dimensional height field to help explain how the method works. The geometry to the right of the point which is highlighted in Figure 2.6 can be visualized as a 1-dimensional height field as shown in Figure 2.7. This example is not completely accurate but is being used to explain the concept.

Figure 2.7 is a mapping of the highlighted part of Figure 2.6. As you can see there is some curvature around the point due to the shape of the curtain. The geometry to the left of the point is closer to the camera as can be seen by the shape of the height field in Figure 2.7. Next three vectors need to be found for the calculation: the normal to P so we can find the tangent to P, and the vector from P to the highest point in the graph. The normal vector and tangent to the point are shown in Figure 2.8.

The normal vector is used to find the tangent, and the vector to the highest point needs to be found

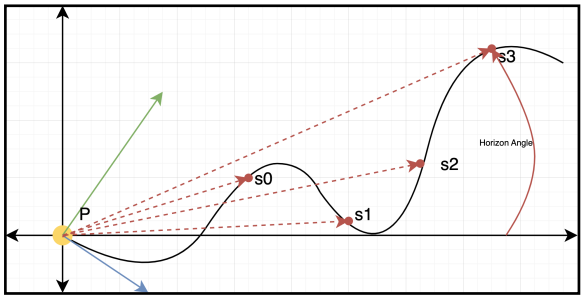


Figure 2.9 1-dimensional height field with sampled points (red dots) and the found horizon angle (solid red line)

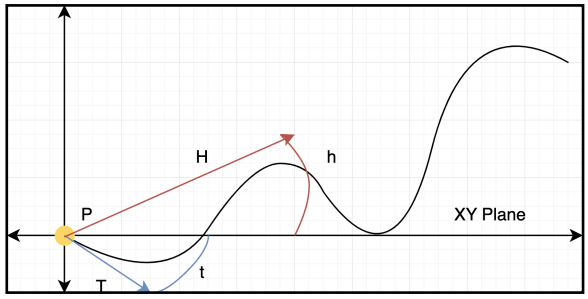


Figure 2.10 1-dimensional height field with tangent vector (T), tangle angle (t), horizon vector (H) and horizon angle (h)

next. Bavoli and Sainz's (2008) approach to this was to 'march on the height field' and check the horizon angle for multiple points on the field then find the largest angle. The horizon angle refers to the angle between a point of observation and the visible horizon or between a point and an object relative to the observer's line of sight to the horizon. In this case, it would be the angle between a sampled point and the x-axis.

Figure 2.9 showcases 4 sample points used to find the horizon angle. Sample s3 ends up giving the largest angle so that is set as the horizon angle. The vector to s3 is also set as the vector pointing to the highest point. The two vectors shown in Figure 2.10 will be used to calculate the occlusion factor. The horizon angle is the angle between the red vector and the x-axis. The tangent angle is the angle between the blue vector and the x-axis. Below the equations demonstrate how to calculate these angles:

$$h(H) = \arctan\left(\frac{H.z}{||H.xy||}\right)$$

$$t(T) = \arctan\left(\frac{T.z}{||T.xy||}\right)$$

Where H and T represent the horizon vector and tangent vector respectively. $H.z$ and $T.z$ their z coordinates. $H.xy$ and $T.xy$ refers to the projection of the respective vectors onto the XY-plane thus $||H.xy||$ and $||T.xy||$ can be calculated as $\sqrt{H.x^2 + H.y^2}$ and $\sqrt{T.x^2 + T.y^2}$ respectively. This gives us the length of the vector in the XY plane. By using arctan we can get the angles h and t. The ambient occlusion can then be calculated using the following equation:

$$AO = sin(h) - sin(t)$$

This would be only for the one direction that was sampled, refer to Figure 2.6, so this would need to be done in multiple directions. NVIDIA's example implementation used 4 directions per pixel but would have the directions randomly rotated. Another aspect to be wary of is the radius of the sampled area for each pixel as this will impact the AO calculation. Larger radii may result in inaccurate occlusion whereas too small a radii may not include geometry that is potentially occluding light. HBAO provides a significant improvement in visual realism compared to simpler SSAO methods. By calculating the horizon angle for each pixel, HBAO effectively simulates the way light interacts with the geometry of the scene, resulting in shadows that accurately reflect the scene's structure.

2.5 Alchemy Screen Space Ambient Occlusion

McGuire et al.'s (2011) Alchemy AO is a more robust derivation of Screen Space Ambient Occlusion which performs typical ambient occlusion operations along with the addition of four parameters: world-space radius, bias, aesthetic intensity, and contrast. All of these are implemented intuitively for the developer to adjust the ambient occlusion to their liking. Similarly to the previously explored SSAO methods, Alchemy SSAO calculates ambient occlusion at a pixel using sample points from depth and normal buffers.

This method aims to address the drawbacks of previous AO methods that were unable to fulfill all of the following requirements: Robust, Mutliscale, Artist-control, and Scalable. Although Alchemy AO still has limitations like sample variance and under-obscurance. The main improvement of Alchemy AO is its function for calculating occlusion which has easily adjustable parameters each with a clear purpose and impact on the function and a built-in falloff function that resembles Filion and McNaughton's Starcraft II SSAO occlusion function (See Figure 2.4). Before the algorithm and its derivation are discussed the sampling method will be explored.

The sampling method used in Alchemy AO is similar to the one used for Volumetric Obscurance (Loos and Sloan, 2010). This method involves selecting samples from a disk with a radius r around the point C, which is parallel to the image plane (See Figure 2.11). Each sample point Q' is chosen uniformly at random within the projection of this disk onto the screen space. The depth or position buffer is then used to find the corresponding camera-space scene point P along the ray passing through Q'. This approach follows the technique proposed by Loos and Sloan (2010), with the understanding that projecting a 3D ball onto the 2D screen results in an ellipse rather than a perfect disk. Although this introduces a small bias in the sampling, the error is considered negligible for typical fields of view up to 70 degrees.

Now to look at the algorithm, McGuire et al. (2011) uses the net incident radiance function which is the sum of near-field radiance and far-field radiance modulated by visibility (Arikan et al. 2005; McGuire 2010). This is used to combine near and far field illumination estimates from different algorithms. The ambient occlusion function from (Landis, 2002) and (Cook and Throne, 1982) is used

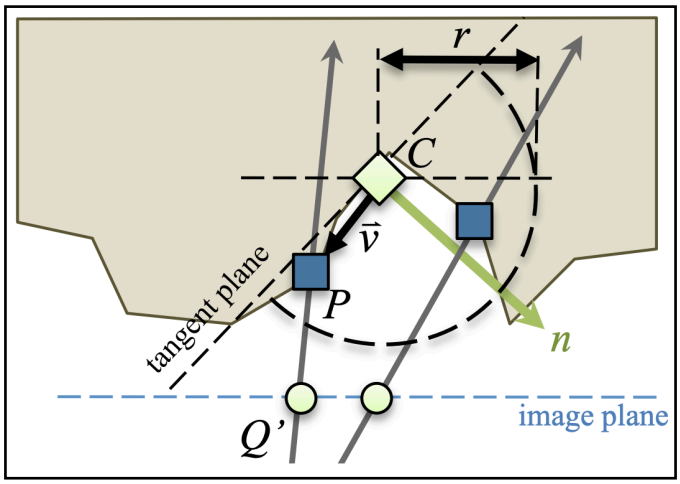


Figure 2.11 Geometric Construction of Alchemy SSAO (McGuire et al., 2011)

and extended to Zhukov et al.'s (1998) ambient obscurance equation using a falloff function resulting in the following equation. (McGuire et al., 2011)

$$A_r(C,\hat{n}) = 1 - \int_{\Omega} \left[1 - V\left(C + \hat{\omega} \cdot \min(t(C,\hat{\omega}) + \epsilon, r)\right)\right] \cdot g(t) \cdot (\hat{n} \cdot \hat{\omega})\, d\hat{\omega}$$

$A_r(C,\hat{n})$ represents the ambient obscurance at point C with surface normal $\hat{n}$. Ω denotes the hemisphere centered around the normal $\hat{n}$. $V\left(C + \hat{\omega} \cdot \min(t(C,\hat{\omega}) + \epsilon, r)\right)$ is the visibility function, which equals 1 if the point along the ray from C in the direction $\hat{\omega}$ is not occluded and 0 if it is. $\hat{\omega}$ is the distance from C to the first occluder along the direction $\hat{\omega}$. ϵ is a small positive value added to avoid dividing by zero. r is the maximum radius considered for near-field occlusion. $g(t)$ is the falloff function, which reduces the influence of occlusion based on the distance t.

Next let $\Gamma \subseteq \Omega$ represent the subset of the hemisphere where there exists a distance $0 < t(C,\hat{\omega}) < r$ such that the ray originating from C in the direction $\hat{\omega}$ intersects the scene. Γ encompasses all near-field occluders projected onto the unit sphere. Since Γ includes all these near occluders, the visibility function from Equation 1 is always zero within this region. (McGuire et al, 2011)

$$V\left(C + \hat{\omega} \cdot \min(t(C,\hat{\omega}) + \epsilon, r)\right) = 0\,|_{\hat{\omega} \in \Gamma}$$

This changes the integration domain in Equation 1 to Γ gives the following equation:

$$A = 1 - \int_{\Gamma} g(t(C,\hat{\omega}) \cdot \hat{n} \cdot \hat{\omega}\, d\hat{\omega}$$

The falloff function that was chosen was $g(t) = u \cdot t \cdot \max(u,t)^{-2}$, as it resembles Filion and McNaughton's Starcraft II occlusion falloff function. This can be substituted in. (McGuire et al, 2011)

$$A = 1 - \int_{\Gamma} \frac{u \cdot t(C,\hat{\omega})}{\max(u, t(C,\hat{\omega}))^2} \cdot (\hat{\omega} \cdot \hat{n})\, d\hat{\omega}\,.$$

Next rewrite this expression using the vector $\mathbf{v}(\hat{\omega}) = \hat{\omega} \cdot t(C, \hat{\omega})$, which represents the vector from point C to the occluder P as shown in Figure. Since $\hat{\omega} = \dfrac{\mathbf{v}(\hat{\omega})}{\|\mathbf{v}(\hat{\omega})\|}$ and $t(C, \hat{\omega}) = \|\mathbf{v}\|$, the equation can be expressed in terms of $\mathbf{v}(\hat{\omega})$. (McGuire et al., 2011)

$$A = 1 - u \int_{\Gamma} \frac{\mathbf{v}(\hat{\omega}) \cdot \hat{n}}{\max\left(u^2, \mathbf{v}(\hat{\omega}) \cdot \mathbf{v}(\hat{\omega})\right)} \, d\hat{\omega}$$

Approximate this integral numerically by randomly sampling a set $\{\hat{\omega}_i\}$ of s directions uniformly resulting in: (McGuire et al., 2011)

$$A \approx 1 - \frac{2\pi u}{s} \sum_{i=1}^{s} \frac{\max(0, \mathbf{v}_i \cdot \hat{n}) \cdot H(r - \|\mathbf{v}_i\|)}{\max(u^2, \mathbf{v}_i \cdot \mathbf{v}_i)}.$$

$H(x)$ is the Heaviside step function.

Next, they introduce adjustable parameters for intensity scale (σ) and contrast (k) to allow for intentional adjustments in the level of occlusion. The intensity is scaled by $1/\pi$, so that when $\sigma = 1.0$ and $k = 1.0$, the result serves as a standard baseline. (McGuire et al., 2011)

u in equation interferes with setting σ independently, and is only used to avoid division by zero. So $\max(u^2, \mathbf{v}_i \cdot \mathbf{v}_i)$ in denominator can get replaced with $\mathbf{v}_i \cdot \mathbf{v}_i + \epsilon$. Here ϵ acts as a small value to avoid division by zero and help prevent underflow and light leaks on corners. Values within two orders of magnitude of $\epsilon = 0.0001$ were able to produce indistinguishable results. (McGuire et al. 2011)

The final addition to the equation is the introduction of a bias distance β which acts as the ambient obscurance analog of shadow map bias. It is scaled by z to adjust for the dot products increasing sensitivity to error in $\hat{n}$ at far points. Resulting in the final equation: (McGuire et al., 2011)

$$A \approx \max\left(0, 1 - \frac{2\sigma}{s} \cdot \sum_{i=1}^{s} \frac{\max(0, \mathbf{v}_i \cdot \hat{n} + z_C \beta)}{\mathbf{v}_i \cdot \mathbf{v}_i + \epsilon}\right)^k$$

Alchemy AO provides an ambient occlusion that is efficient with its sampling and calculations providing visuals that improve on previous methods and introducing easily adjustable parameters for artists to tweak and improve the visuals to their liking.

2.6 Deferred Rendering

Deferred rendering provides an efficient framework for Screen Space ambient occlusion methods. By separating the geometry and lighting stages, deferred rendering allows for the collection of essential data, such as depth and normals, into a G-buffer, which is crucial for accurate AO calculations.

It is based on the concept of postponing the majority of the heavy rendering to a later stage. The processing of a scene is divided into two distinct stages: the geometry pass and the lighting pass. In the geometry pass the scene's geometric information such as depth, surface normals, and material properties is captured and stored in multiple buffers, collectively known as the G-buffer. Next, in the lighting pass, lighting calculations are applied to the data stored in the G-buffer, efficiently handling complex lighting scenarios. It is particularly useful for scenes with many lights, as it decouples the number of lighting calculations from the complexity of the scene, improving performance and

enabling more advanced visual effects (De Vries, 2020). Figure 2.12 taken from De Vries' (2020) LearnOpenGL page on deferred shading gives a clear picture of the deferred shading process.

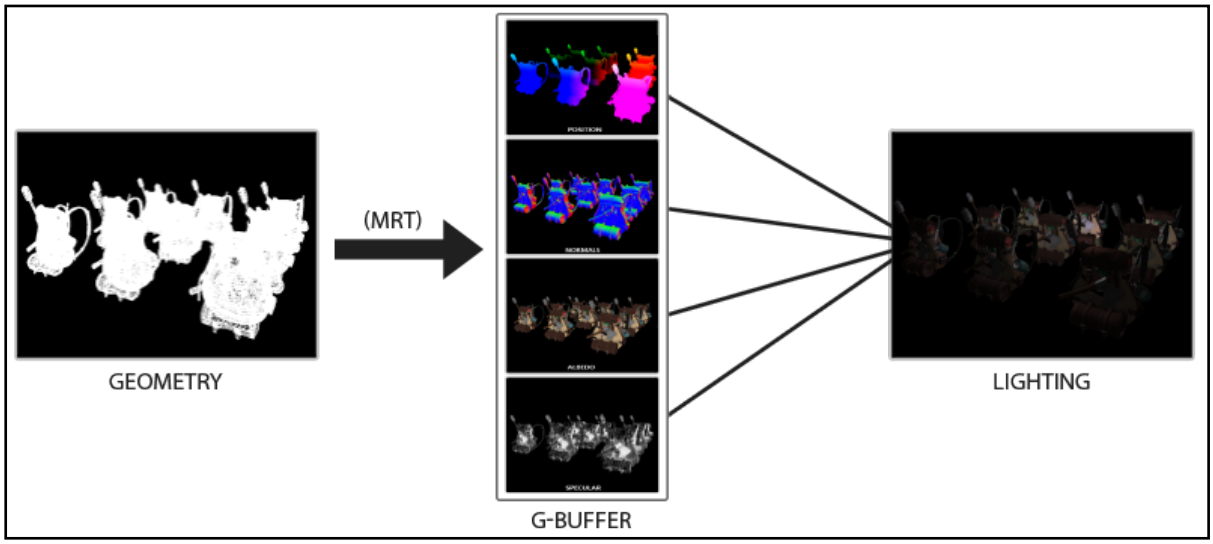


Figure 2.12 Illustration of defferred rendering process (De Vries, 2020)

2.7 Requirements

Having researched the Ambient Occlusion methods to be implemented, the requirements can be set for the project now. The table below covers the requirements to sufficiently and effectively compare each Starcraft II's SSAO, HBAO, and Alchemy SSAO. Requirements are categorized into non-functional and functional.

Requirement	Functional or non-functional
The application can load a scene with a model, lighting and a camera setup.	Functional
The code for the application is well structured and organized in an easy to understand manner.	Non-functional
The code is well commented.	Non-functional
The application has Starcraft II's SSAO, HBAO and Alchemy AO implemented.	Functional
Users can switch between the three different ambient occlusion settings or turn off ambient occlusion altogether.	Functional
Users can switch between preset camera angles or move around themselves to see how the AO effect looks from different angles.	Non-functional
Users are provided a GUI which shows which AO setting is currently on	Non-functional

Requirement	Functional or non-functional
Users are provided a GUI allowing live adjustments of AO parameters	Functional
Users can toggle the GUI and lock or unlock camera movement to access the GUI	Non-functional
The application should display the current Frames per second (FPS) and camera position	Non-functional
The application should have a test mode that switches between camera angles and AO settings and logs the average FPS for each setting	Functional

Chapter 3
Methods

3.1 Design Overview

This section will go over the planned workflow to meet the project requirements. Each step will be looked at in more detail in this chapter.

1. **Graphics application setup:**
   An application setup that can load models, and has a working camera and lighting setup. Set the foundation to implement and compare all three ambient occlusion methods within the same environment.
2. **Starcraft II's Screen Space Ambient Occlusion implementation:**
   Implement this SSAO method into the graphics application along with a deferred shading setup, also setting up the rendering pipeline for the other two ambient occlusion methods.
3. **NVIDIA's Horizon-Based Ambient Occlusion implementation:**
   Implement HBAO into the graphics application. Requires the addition of the HBAO fragment shader and adjustments to the main application code.
4. **Alchemy SSAO implementation:**
   Implement Alchemy SSAO into the graphics application. Requires the addition of the Alchemy SSAO fragment shader and adjustments to the main application code.
5. **Camera and testing setup:**
   Implement preset cameras and an automatic testing setup that tests each method at different camera angles in a scene and logs the performance (in frames per second) for each.
6. **Additional features:**
   Throughout the process implement additional useful features such as a GUI and adjustable AO parameters to test and tweak each method. Any other quality-of-life features will also be covered here.

3.2 Graphics Application Framework

The framework for the graphics application used in this project is based on De Vries' (2020) LearnOpenGL tutorial. This tutorial provides us with an OpenGL graphics application that uses additional technologies GLFW, GLM, Assimp, and C++ code. GLFW is used for window management, GLM is a standard mathematics library, Assimp is an open asset import library used for loading and processing 3D models, and C++ is the programming language used for this application.

Below are the features provided in De Vries' (2020) application from the tutorial:

- Create an OpenGL context and an application window
- Setup shaders
- Load textures
- Load models using Assimp
- Setup a camera that can move around the scene
- Setup basic lighting

This application setup provides us with a foundation for the implementations. The first action to be taken after setting up the application is to choose and load the scene that will be used to compare the AO methods. The model chosen for testing is the industry standard Crytex Sponza model from McGuire's Computer Graphics Archive (2017). Originally released by Crytek, it has become a popular choice for graphics development testing due to its complex geometry, diverse materials, challenging lighting conditions, and reputation for pushing performance limits.

3.3 Starcraft II's Screen Space Ambient Occlusion Implementation

For the SSAO implementation De Vries' (2020) LearnOpenGL tutorial also includes an SSAO implementation based on Starcraft II's SSAO (Filion and McNaughton, 2008). This was used as a base for this project's implementation. This section will cover the practical aspects of de Vries' (2020) implementation based on the concepts covered in section 2.3.

Sample Buffers

As this implementation is a Screen Space method, it requires geometrical information to calculate the occlusion factor for each fragment. So de Vries (2020) states the following data is needed:

- A per-fragment position vector
- A per-fragment normal vector
- A per-fragment albedo color
- A sample kernel
- A per-fragment random rotation vector to rotate the sample kernel

For this reason, the SSAO is implemented on top of a deferred renderer. De Vries (2020) sets up the fragment shader of the geometry stage to store the fragment position vector, the per-fragment normals, and the diffuse per-fragment color into the gbuffer texture. In the main application code, a texture is configured that stores the position data, normal data, and color and specular data for each fragment on the screen. These textures will be used in calculations for each AO method.

Sample Kernel

A sample kernel needs to be generated for the shader. For Starcraft II's SSAO method, the sample kernel needs to be a normal oriented hemisphere. To do this De Vries (2020) generates a single sample kernel in tangent space (local coordinate space) with the normal pointing in the positive z direction. This sample kernel can be transformed easily to align with the normals of each fragment when calculating the occlusion using the surface's normal vector.

To generate the kernel De Vries (2020) assumes a unit hemisphere (the actual hemisphere scale can be adjusted in the shader) and varies random x and y values between -1.0 and 1.0 and z values between 0.0 and 1.0. These coordinates should all fall within a unit hemisphere with a normal pointing in the positive z direction. To improve the sampling an accelerating interpolating function is used so that the kernel distributes more samples closer to the origin.

The last addition to the sampling done by De Vries (2020) is the addition of random kernel rotations to reduce the number of required samples to get good results. As discussed in section 2.3 Fillion and McNaughton (2008) recommend the use of a texture of random rotation vectors. To do this De Vries (2020) creates a 4x4 array of random rotation vectors oriented around the tangent-space surface normal and stores them in a 4x4 texture that is tiled over the screen. This results in a sample kernel with random rotations for improved sampling.

SSAO Shader

The sample buffer and sample kernel setup give us the required input data for the SSAO shader. To store the result of the SSAO another framebuffer object is created and is processed between the geometry pass and the lighting pass. The SSAO shader takes the previously set G-buffer textures, the noise texture (random rotation vectors), and the kernel samples. With the fragment position data and the normal data that we get from the textures, De Vries (2020) creates a TBN matrix that transforms vectors from tangent space to view space. The Gramm-Schmidt process creates an orthogonal basis (TBN matrix) for shading, using a random vector to slightly tilt the basis each time. This adds controlled randomness, removing the need for exact alignment with the surface, and eliminates the need for per-vertex tangent and bitangent vectors, simplifying calculations while maintaining effective shading. (De Vries, 2020).

The next step in the process is to iterate over each sample in the kernel and transform them from tangent to view space, add them to the current fragment's position, and compare their depths.

```
vec3 samplePos = TBN * samples[i];

samplePos = fragPos + samplePos * radius;
```

The *radius* is a variable that can be tweaked to our liking. De Vries (2020) by default sets its value to 0.5. After tweaking the radius in the scene set up for this project, a radius value set to 1.3 provided the best outcome. Each sample will be transformed to view space and added to the fragment position, then multiplied by the radius to adjust the sample hemisphere size of the SSAO (De Vries 2020).

Next, the sample is converted from view space to clip space using the projection matrix. The resulting vector goes through a perspective divide, a process that converts coordinates from clip space to normalized device coordinates. These coordinates are further adjusted to fit within the [0.0, 1.0] range, making them suitable for sampling the position texture (De Vries, 2020).

```
vec4 offset = vec4(samplePos, 1.0);

offset      = projection * offset;    // from view to clip-space

offset.xyz /= offset.w;               // perspective divide
```

```
offset.xyz  = offset.xyz * 0.5 + 0.5; // transform to range 0.0 - 1.0
```

Once transformed, the x and y components of the offset vector are used to sample the position texture, retrieving the depth value of the sample as seen from the camera's perspective. The sampled depth is then compared to the original sample's depth. If the sampled depth is greater, indicating that the sample is behind the geometry, an occlusion factor is added. A small bias is applied to the depth comparison to prevent visual artifacts.

De Vries (2020) implements a range check to ensure that a fragment contributes to the factor only if its depth value falls within the sample's radius. The implementation uses GLSL's 'smoothstep' function, which smoothly interpolates between 0.0 and 1.0 based on the depth difference and the radius. If the depth difference is within the radius, the function returns a value between 0.0 and 1.0.

Finally, the occlusion contribution is normalized by the size of the kernel and subtracted from 1.0, directly scaling the ambient lighting of each fragment.

```
float rangeCheck = smoothstep(0.0, 1.0, radius / abs(fragPos.z -
sampleDepth));

occlusion += (sampleDepth >= samplePos.z + bias ? 1.0 : 0.0) * rangeCheck;
```

Blur

Bur needs to be applied between the SSAO pass and the lighting pass. Another framebuffer object is made for storing the blur result in De Vries's (2020) implementation. The tiled random vector texture already gives a consistent randomness that can be used to create a simple blur shader.

3.4 NVIDIA's Horizon-Based Ambient Occlusion Implementation

To implement the HBAO another framebuffer for the HBAO and blur needs to be added, as well as the HBAO fragment shader which contains the occlusion calculation. The framebuffers were set up in the main application code similarly to the SSAO framebuffer and adjustments were made to the rendering pipeline. Preemptively the framebuffers for Alchemy SSAO were also set up. All three methods were placed in the rendering pipeline between the geometry pass and the lighting pass. Each behind a check for whether they were enabled or not. Below is a diagram visualizing the rendering pipeline.

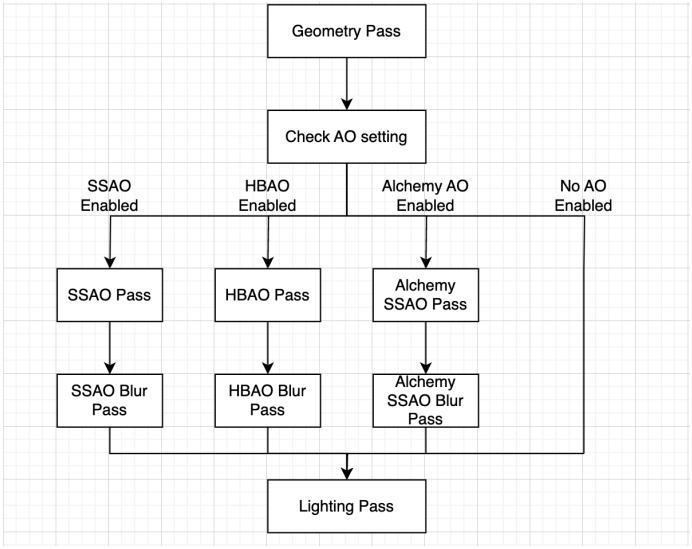


Figure 3.1 Rendering pipeline for the application

HBAO Shader

For the HBAO fragment shader meemknight's (2023) OpenGL HBAO implementation was used as a base for the code. Meemknight's (2023) 'computeAO' function is designed to calculate part of the HBAO effect for a given fragment. It is called with a fragment position and a direction and performs ray marching to calculate the occlusion on that point. This function will be called four times for four directions on each fragment.

First, the radius is calculated. It scales with the fragment's depth so closer objects will have a larger on-screen radius than further objects. This calculated radius is then clamped between a minimum and maximum radius. After testing and comparing many radius values, a value of 500,000 with clamps of 3 and 100,000 gave the best outcome. The way the radius is scaled makes it require a large initial value.

Next, a local coordinate system was established by calculating the viewing direction and a perpendicular axis relative to the specified sampling direction. It then projected the surface normal onto a plane defined by this system, simplifying the problem to two dimensions and allowing for more efficient occlusion calculations. By computing the tangent vector and determining its angle relative to the XY-plane, the necessary geometric relationships to evaluate potential occlusions were set up. Additionally, it prepared the parameters for ray marching, including consistent screen-space sampling, and initialized variables to track the maximum depth and corresponding position encountered during the occlusion evaluation process.

With the required set up complete ray marching can be performed next. A loop is initialized to iterate through 4 samples as this function will be called four times in different directions. Giving a total of 16 samples per fragment which is equal to the amount of samples used for the earlier SSAO implementation. Allowing for both methods to be compared with the same number of samples. For each iteration, a new march position in texture coordinates is calculated. The position of the fragment at the march position is sampled and a horizon vector is computed by normalizing the march position with the original fragment position. This vector represents the direction from the original fragment to the sampled point and is used to determine if the sampled point is above or below the horizon of the original fragment. A check is then performed to determine if the z-component of the horizon vector is the highest encountered so far. If it is, the highest z-value is updated, and the position of the fragment corresponding to this highest z-value is saved.

With the horizon point found, the occlusion contribution can be calculated. Below are the calculations done in pseudocode.

```
horizon_vector = horizon_point - fragment_position

horizon_angle = arctan(horizon_vector.z / length(horizon_vector.xy))

sin_horizon_angle = sin(horizon_angle)

result = (sin_horizon_angle - sin_tangent_angle)/2
```

The result is also clamped between a value of 1.0 and 0.0. The result is then returned from the 'computeAO' function.

In the main section of the shader, the function is called 4 times in a cross. A random 2D vector is chosen as a direction and the opposite direction, and the two perpendicular directions are chosen. For each fragment, the 'computeAO' function is called 4 times with those directions and the fragment position as parameters. The results returned are aggregated and divided by the sum of the lengths of the projected normals. The result is then multiplied by a darkness factor, added to increase the

intensity of the occlusion effect. Finally, this value is subtracted from 1.0 adjusting the brightness of every fragment in the scene.

3.4 Alchemy Screen Space Ambient Occlusion Implementation

To implement McGuire et al.'s (2011) Alchemy SSAO only a fragment shader needs to be made, as the framebuffer and pipeline are already set up for this implementation. To begin the implementation De Vries' (2020) SSAO fragment shader was used as it gave a foundation with a noise texture and normal and position view-space conversions. However, the sampling and ambient occlusion calculation still needs to be added.

Diskpoint Sampling

To implement Diskpoint sampling a function is created that takes a radius, two random values and a turns parameter as inputs and generates a 2D point on the disk. The random values are generated using the following code:

```
vec2 RandomHashValue(float randomValue)
{
    return fract(sin(vec2(randomValue, randomValue + 0.1)) * vec2(12.9898,
      78.233));
}
```

This is a commonly used pseudo-random number generator used in many graphics applications. Below is the Diskpoint function based on Spektre's (2022) function:

```
vec2 DiskPoint(float sampleRadius, float x, float y, float turns)
{
    float r = sampleRadius * sqrt(x);
    float theta = y * (2.0 * PI) * turns;
    return vec2(r * cos(theta), r * sin(theta));
}
```

The radius is scaled by the square root of 'x' to ensure a uniform distribution of points across the disk. The angle 'theta' is determined by the 'y' value and the 'turns' parameter, which controls how tightly the points are spiraled around the disk's center.

In the main section, the radius input to the function is scaled with the fragment depth. Thus further fragments have smaller radii. Once the function is called, the returned sample position is used. The vector from the fragment position to the sample position is found and the length of it is calculated. Now the Alchemy SSAO calculation can be performed. The results for each sample are accumulated for each fragment. Below is the equation calculated in the loop and the respective code.

$$\sum_{i=1}^{s} \frac{\max(0, \mathbf{v}_i \cdot \hat{n} + z_C \beta)}{\mathbf{v}_i \cdot \mathbf{v}_i + \epsilon}$$

```
float occlusionFactor = max(0.0, dot(V, normal + fragPos.z * beta)) /
(dot(V, V) + epsilon);
ao += occlusionFactor;
```

After the loop, the rest of the equation is completed and the fragment outputs the Alchemy Ambient Occlusion value calculated.

$$A \approx \max\left(0{,}1 - \frac{2\sigma}{s} \cdot \sum_{i=1}^{s} \frac{\max(0, \mathbf{v}_i \cdot \hat{n} + z_C\beta)}{\mathbf{v}_i \cdot \mathbf{v}_i + \epsilon}\right)^k$$

```
ao = max(0.0, 1.0 - (2.0 * sigma / float(kernelSize)) * ao);
ao = pow(ao, float(k));
```

Alchemy SSAO has many parameters that need to be tweaked for optimal visuals. A sample size of 16 was used, to equal the sample size of the other two methods. After lots of experimenting, the settings that produced the best results and were used for testing were the following:

Samples = 16

Radius = 1.7

Bias = 0.025

σ (Strength Multiplier) = 1.7

k (Contrast Multiplier) = 1

β (Shadow Bias) = 0.5

Turns = 10

ϵ (Value to avoid division by zero) = 0.001

3.5 Camera and Testing Setup

A structured testing framework has been implemented within the application to evaluate the performance and visual impact of different ambient occlusion techniques. This framework is designed to measure and compare the effects of SSAO, HBAO, Alchemy SSAO, and scenarios, with no ambient occlusion applied. The primary objective is to gather empirical data on frame rates and to conduct a visual analysis through screenshots, facilitating a comprehensive comparison of each method's impact on graphical fidelity and performance.

Camera Angles: The application utilizes four predetermined camera angles to provide unique perspectives and compositions of the scene Each angle is selected to ensure a comprehensive evaluation across different object densities, depth variations, and spatial complexities.

Ambient Occlusion Settings:

Four settings are tested: SSAO, HBAO, Alchemy SSAO, and no occlusion. Each setting is applied continuously for five seconds per camera angle.

Testing Cycles: To ensure the reliability of the data, the application cycles through each occlusion setting three times for each camera angle. This repetition helps mitigate anomalies due to transient system or environmental factors.

Frame Rate Measurement: Frame rate (FPS) is recorded during the testing of each setting. Higher frame rates indicate better performance, suggesting a lighter computational load. Lower frame rates suggest a heavier computational load, potentially impacting performance in resource-constrained scenarios.

Visual Documentation: Screenshots are captured at each camera angle under each ambient occlusion setting to facilitate side-by-side visual comparisons.

Performance and Visual Analysis: The systematic approach helps quantify the performance impact and qualitatively evaluate the visual enhancements each technique offers. The results aim to provide insights into selecting an appropriate ambient occlusion technique based on performance requirements and visual fidelity goals.

3.6 Additional Features

This section will cover additional features added to the application outside of the Ambient Occlusion implementations.

Directional Lighting

Directional lighting was added to the scene to simulate the effect of light coming from a distant source like the sun. It helped brighten the scene and test it in an environment like a video game scene.

Graphical User Interface

A Graphical User Interface (GUI) was essential to the application. For this application, the GUI contained the following:

- Framerate
- Camera Position
- Button to toggle SSAO
- Button to toggle HBAO
- Button to toggle Alchemy SSAO
- Button to toggle Textures
- SSAO Parameter sliders
- HBAO Parameter sliders
- Alchemy SSAO Parameter sliders.

These allowed for clarity to the user over what state the scene was currently in and allowed for adjustments of parameters to test different Ambient Occlusion settings.

Chapter 4
Results

4.1 Application Features

The details of implementation methods used are covered in the previous section. This is an outline of the application and its final features.

- It can load in the Crytex Sponza model
- Users can toggle textures on or off.
- The camera can move around using WASD user input and can zoom and or be sped up
- Preset cameras are also set up which can be cycled through
- SSAO, HBAO, and Alchemy SSAO are implemented and can be toggled on or off (if all are off there is no ambient occlusion applied)
- The GUI shows the SSAO, HBAO, Alchemy SSAO, and texture toggles. It also shows the current FPS and camera position. With sliders to adjust the parameters of each AO method. The GUI can be toggled on or off and the camera can be locked so the cursor can be used for the GUI.

Figure 4.1 Screenshot of application with GUI visible

4.2 Test Results

As explained in section 3.7 the occlusion methods were compared over 4 different camera angles. The test was done with no textures to more clearly see and assess the effect of each occlusion method. To more clearly compare each method, the scene was loaded without textures for the tests. Screenshots of the scenes with textures can be seen in Appendix C. The graphics application was set to run at a resolution of 1920x1080, as this is standard HD. Tests were carried out on a computer with the following specifications.

Operating System: Windows 10 x64

Processor: Intel(R) Core(TM) i5-10400F CPU @ 2.90GHz.

Graphics Processing Unit: NVIDIA GeForce RTX 2070 Super.

RAM: 32.0 GB.

Refresh Rate: 240Hz.

Screenshots

Below are screenshots of the scene rendered with each method at each camera angle.

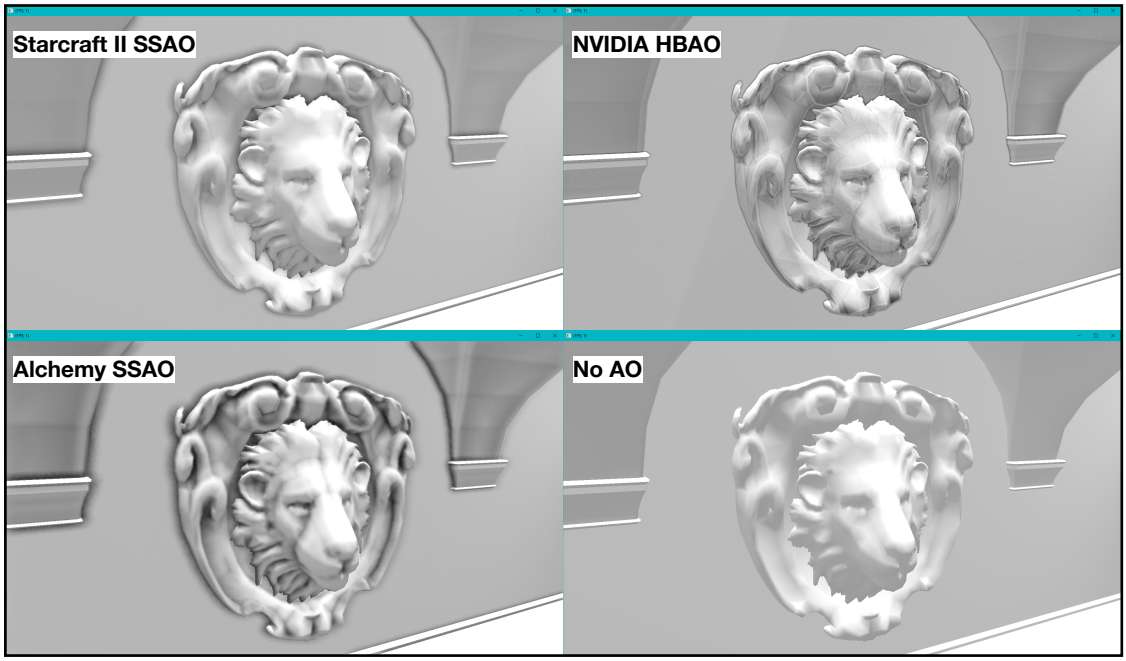


Figure 4.2 Camera Angle 1

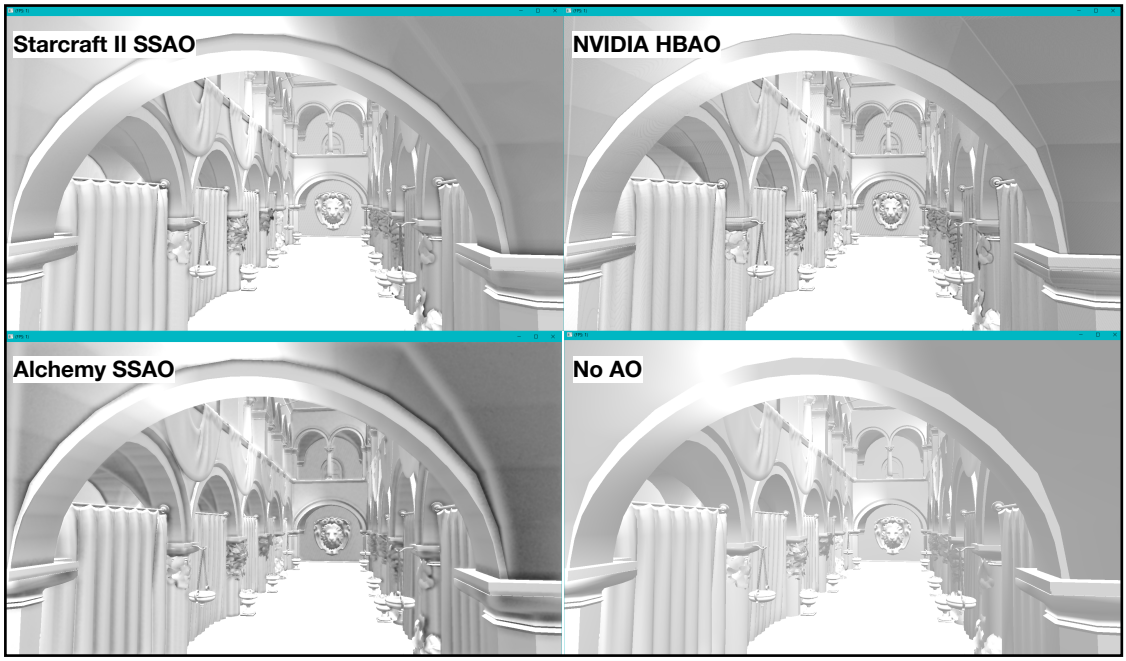


Figure 4.3 Camera Angle 2

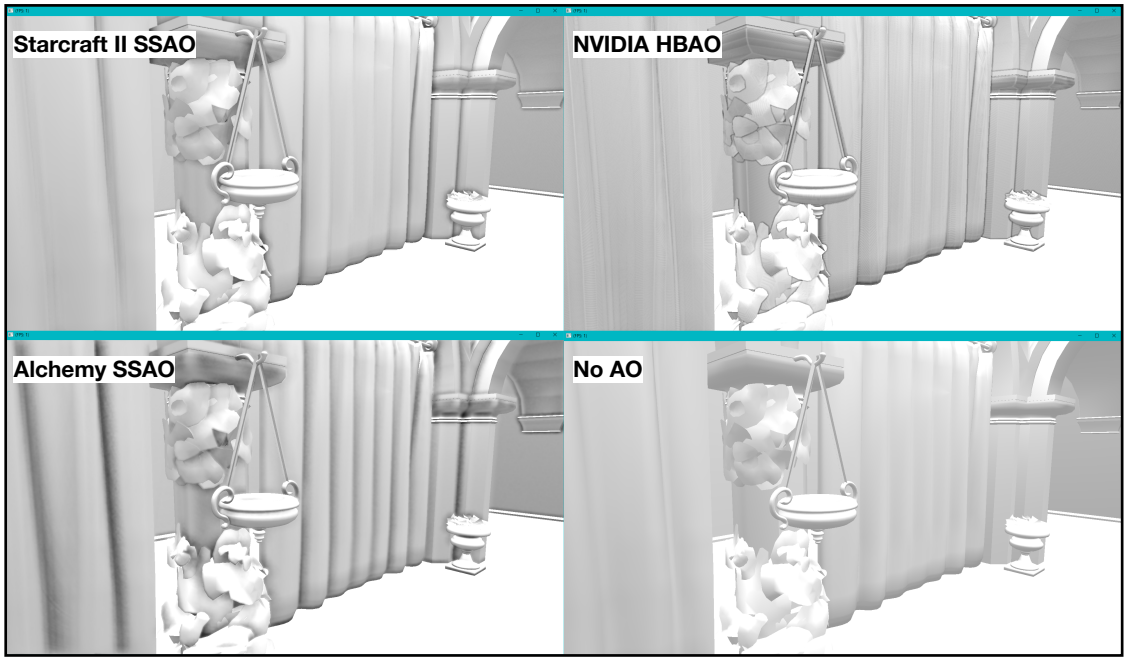


Figure 4.4 Camera Angle 3

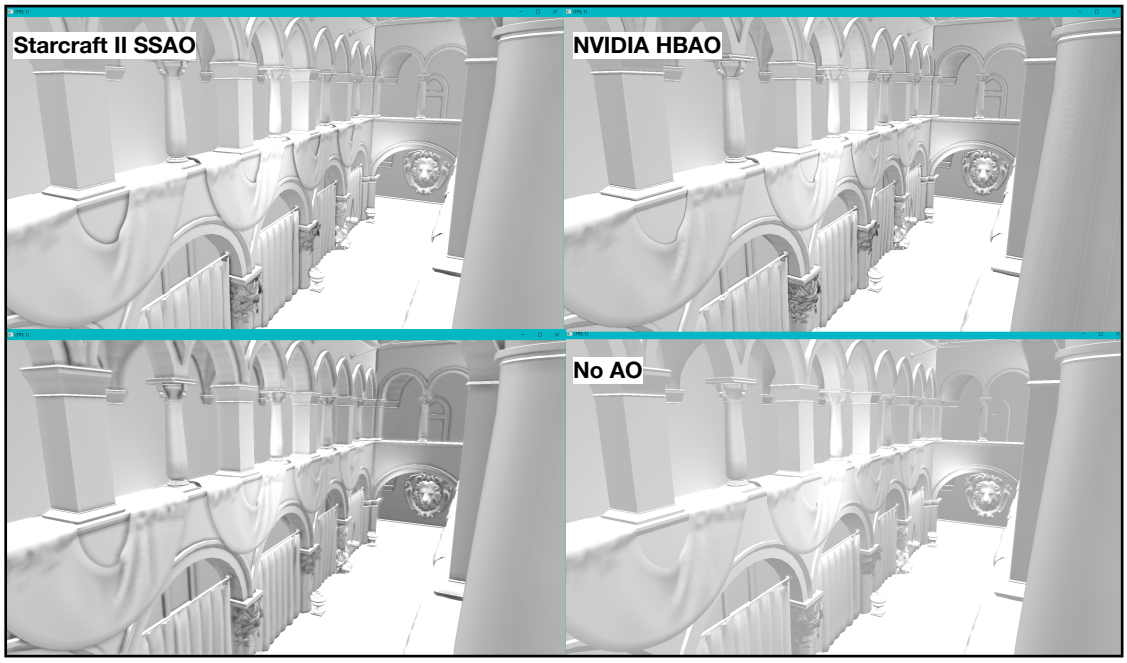


Figure 4.5 Camera Angle 4

Performance

The table below shows the average frames per second measured for SSAO, HBAO, Alchemy SSAO, and no AO for each camera angle over 10 trials. The full trial data can be viewed in Appendix C.1.

Camera	SSAO (Average FPS)	HBAO (Average FPS)	Alchemy SSAO (Average FPS)	No AO (Average FPS)
1	224.427	230.479	226.296	234.366
2	228.891	229.843	221.018	232.881
3	229.905	232.405	224.570	234.586
4	228.823	228.943	224.301	236.112
Average	228.011	230.418	224.046	234.486

Table 1 Average FPS of each technique over 10 trials and 4 different camera angles

Chapter 5
Discussion

5.1 Analysis

A visual comparison will be carried out by analyzing the scenes' screenshots in Figures 4.2, 4.3, 4.4, and 4.5. For every camera angle, a screenshot of each method will be shown. On this screenshot numbers will be placed around the scene in the No AO screenshot. These numbers will reference the areas of the scene to look at during the analysis.

Camera 1

To start we can look at Camera 1 (Figure 5.1). In this position, the camera is placed close to the geometry of a lion head plaque. The geometry depth and shape are clearer with ambient occlusion on. All three ambient occlusion methods greatly improve over having no ambient occlusion. All three methods successfully outline objects and darken crevices, mimicking realistic light interactions. Alchemy SSAO and Starcraft II's SSAO provide similar outcomes with Alchemy SSAO having more intense shadows and increased detail. At position 1, the lion's forehead, Alchemy AO correctly darkens a crease going down the middle of its forehead which neither of the other methods does. Around the edge of the plaque, position 2, are deep crevices. The contrast in the darkness between these shadows and those in the mane is clear in Alchemy AO. In position 3, we can see some banding

occurring in each scene. This artifact should not be present in any of the methods but exists due to implementation mistakes from a poorly set bias or a fault in the blur. Therefore, this will not be taken into account when judging effectiveness. In position 4 and throughout the scene HBAO has very detailed geometry. This can be beneficial for visual clarity but in this case is too much. Both SSAO and Alchemy SSAO have smoother shadows.

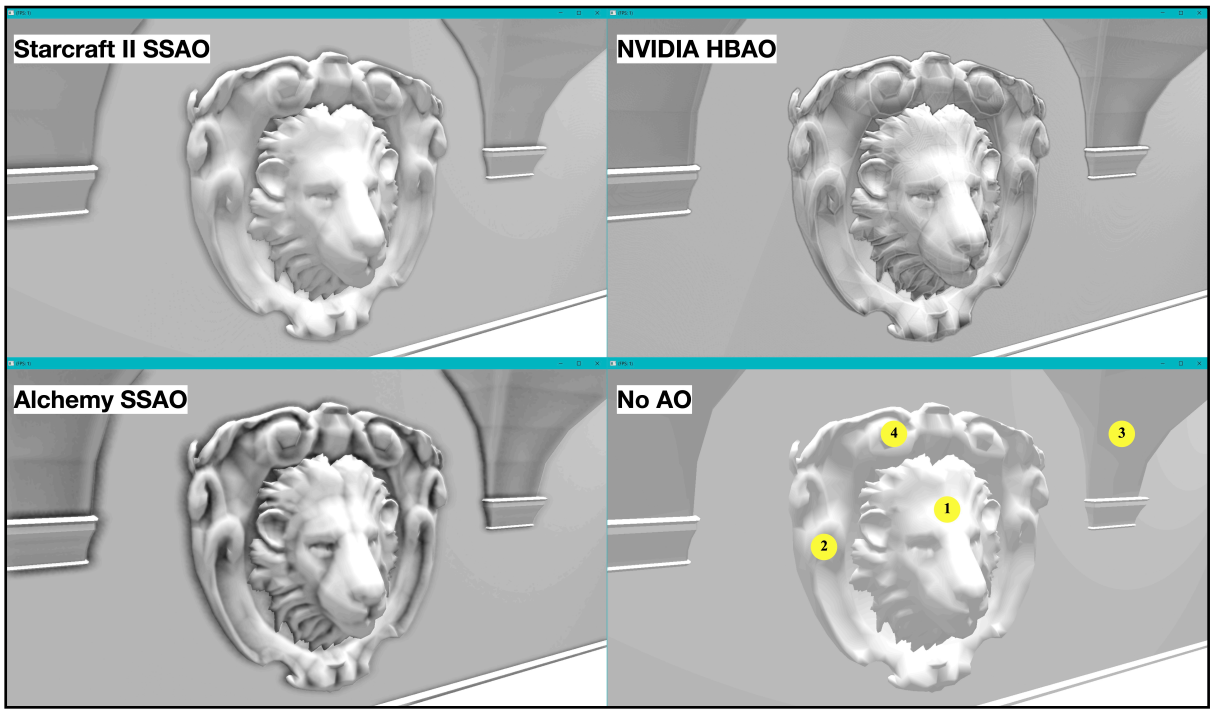


Figure 5.1 Camera 1

Camera 2

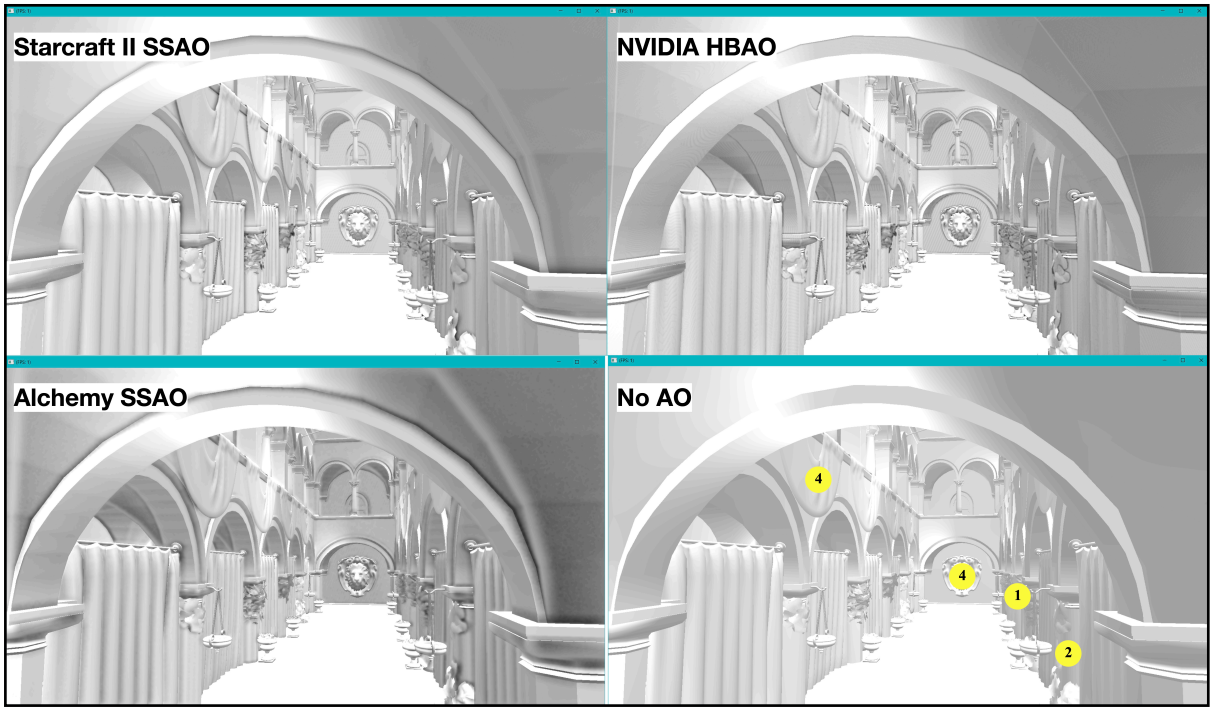


Figure 5.2 Camera 2

The Camera 2 scene looks across a large portion of the model. It includes interesting geometry such as curtains, pots, plants, and the lion plaque from a further distance. Once again all three methods look significantly better than the scene without ambient occlusion. Around position 1 the shadows on the leaves and pots help make the geometry clear. Starcraft II's SSAO and HBAO provide clear shadows and details around the leaves whereas Alchemy AO has smoother but less clear leaves. At position 2, the closest leaves, we can analyze the interaction more closely. HBAO provides a very clear outlining of the geometry and Alchemy AO provides a soft shadow over darker areas. Starcraft II's SSAO is somewhere in between both. In position 3 we see the lion plaque again. HBAO and Alchemy AO apply light shadows to it while SSAO applies very light shadows due to its range check. In position 4 we can see the drapes on the walls. Each method provides some outline so the drape does not blend into the wall like the scene without ambient occlusion. SSAO provides a clear outline and shadow for every drape, HBAO's further drapes have less distinction, and Alchemy AO has unclear outlines but accurate shadows.

Camera 3

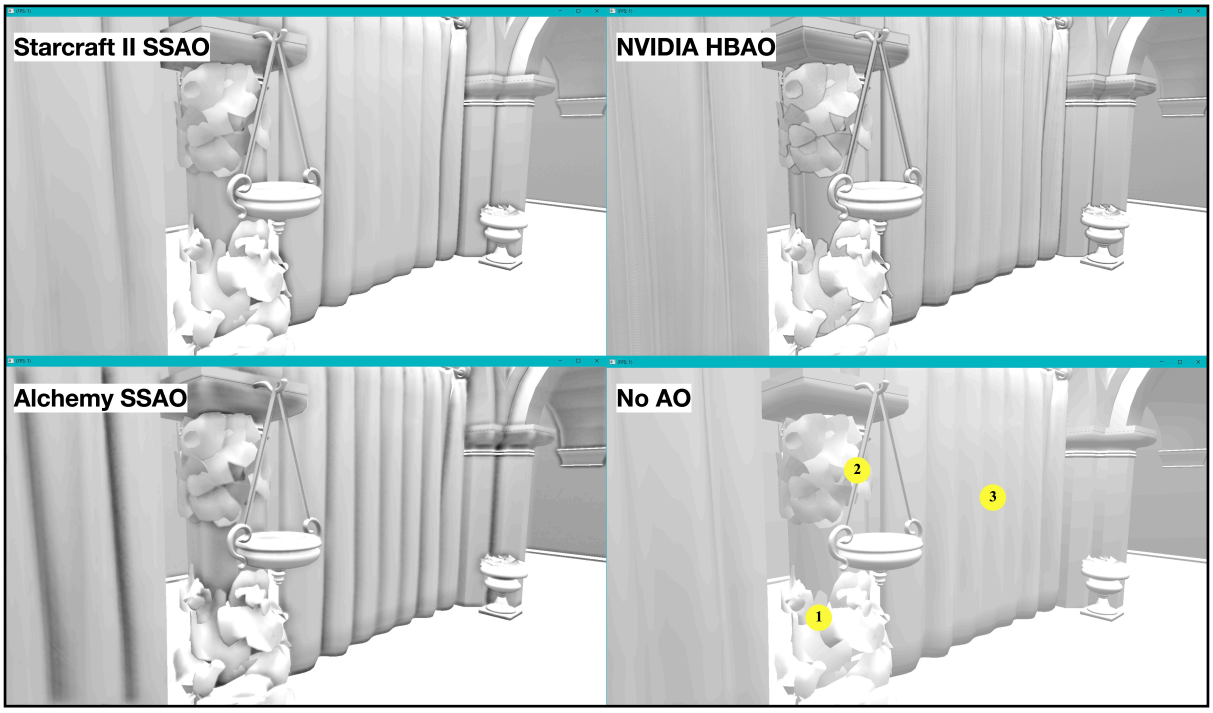


Figure 5.3 Camera 3

Camera 3 positions the camera for a close-up view of a pot with plants, a hanging pot, some leaves on a pillar, and curtains. In position 1, the leaves in HBAO have distinct outlines but lack the depth and shading of SSAO and Alchemy SSAO. Alchemy SSAO also has higher shadow intensity and contrast with clear differences between the darkest portions and lighter shadows. In position 2, the rope holding the pot has an unwanted artifact in the HBAO implementation. The shadows are bleeding outside of the geometry of the rope, which does not occur in the other two implementations. However, this may be a fault in the implementation. In position 3, Alchemy AO and HBAO offer better darkening of the curtain's crevices. Alchemy AO provides an especially realistic-looking darkening of the curtains.

Camera 4

Camera 4 positions the camera on the second floor of the model. It has a better view of the second floor and its arches and drapes. As well as providing a different angle to see how the light and shaders interact with the geometry. In position 1 we once again look at the drapes. From this angle, every method does a better job of outlining each drape. SSAO still provides the most clear outlining with accurate shadows. However, the detail in the drapes is clearer in HBAO and Alchemy AO. In position 2 the leaves have the most detail in HBAO, whereas Alchemy SSAO's is blurrier. Both SSAO and HBAO have more contrast between the dark and light shadows in the leaves from this angle. Additionally, SSAO has little to no banding in the scene. HBAO has light banding throughout the scene and Alchemy SSAO has heavy banding in some areas.

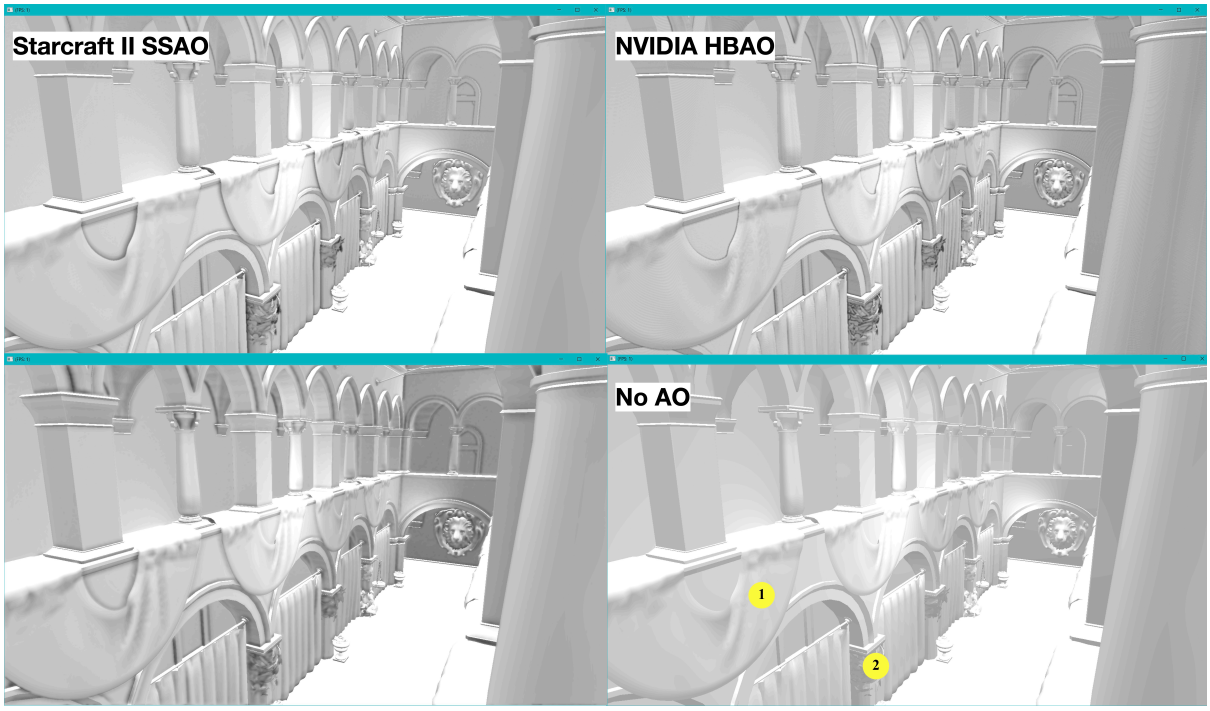


Figure 5.4 Camera 4

Performance Analysis

The results in Table 1 show a clear distinction in the performance of these implementations of Starcraft II's SSAO, NVIDIA's HBAO, and Alchemy SSAO. All three methods were implemented using 16 samples for fair assessment. It is clear that having no occlusion results in the best performance with more than 4 frames per second increase over HBAO, more than 6 over SSAO, and more than 10 over Alchemy SSAO. This is expected as ambient occlusion takes some computational load. Overall HBAO has the best performance, followed by SSAO then Alchemy SSAO. These results are somewhat unexpected as SSAO should be the simplest computation and thus the best performance. However, none of these implementations are perfect and factors such as optimization and implementation details may affect the performance of each.

5.2 Conclusions

The visual comparison of each method shows how effective ambient occlusion is. At every camera angle, ambient occlusion enhanced the realism by adding depth and clarity. HBAO provided finer

details and clear distinctions between geometry, especially in complex areas like the leaves and the lion plaque. SSAO had the fewest details but had clear geometry and dark shadows. Alchemy SSAO provided the most intense shadowing with great detail and contrast but in some areas, shadows were too soft causing reduced geometrical clarity. Performance-wise HBAO ran at a much higher FPS than SSAO or Alchemy SSAO. However, each method performed reasonably well compared to no ambient occlusion. Although the implementations for this application were limited and may not perfectly represent each technique, much can still be drawn from this comparison.

Overall, the choice between each technique will depend on the specific needs and constraints of the project. HBAO provides more precise occlusion details and is generally faster in this implementation, making it well-suited for high-performance applications that require visual accuracy. Whereas, SSAO, despite its less consistent performance, offers stylistically stronger and more pronounced shadows that can be desirable in settings where dramatic lighting is key. Apart from performance, Alchemy SSAO acted as an almost direct improvement over Starcraft II's SSAO. Additionally, the easy-to-use parameters greatly improve the usability and customization of Alchemy AO, making it ideal for artists. Each technique has merits, and selecting between them should be guided by the particular artistic goals and performance considerations of the rendering context. Ultimately the addition of ambient occlusion significantly enhances the visual richness and depth of 3D scenes, contributing substantially to the realism and immersion of each scene. As far as these implementations of these techniques, Alchemy AO provided the best visuals as well as being the most easily customizable effect, although its poorer performance will have to be taken into account.

5.3 Ideas for future work

Algorithm Optimization

Delving deeper into optimizations tailored to each algorithm is essential. For each method, experimenting with various sampling techniques to minimize artifacts while maintaining performance could be interesting. Further refining the algorithm to enhance performance across different hardware platforms, possibly integrating machine learning techniques to dynamically adjust parameters for optimal quality and performance.

Adaptive Sampling

Developing adaptive sampling methods that adjust the number of samples based on the geometric complexity or spatial variance within the scene will help reduce computational costs while ensuring high-quality results where finer details are crucial.

Cross-Platform Performance Analysis

Extending the testing of each method across a broader range of hardware platforms, including mobile devices and consoles, to understand their performance and scalability in varied environments will aid in designing more robust ambient occlusion techniques effective across all platforms.

Comparative Exploration of Other Ambient Occlusion Methods

Similarly to this paper, I could attempt implementing, testing, and comparing other ambient occlusion methods such as Ambient Occlusion Volumes, Voxel-Based Ambient Occlusion, or Ray Traced

Ambient Occlusion. All of these are modern techniques that have improved Ambient Occlusion and would offer interesting comparisons.

List of References

Arikan, O., Forsyth, D.A. and O'Brien, J.F., 2005. Fast and detailed approximate global illumination by irradiance decomposition. *ACM Transactions on Graphics*, 24(3), pp.1108–1114 [Accessed 28 July 2024].

Bavoli, L. and Sainz, M.(2008) 'Image-Space Horizon Based Ambient Occlusion,' 'SIGGRAPH2008, Los Angeles, United States, August 11-15. Available at https://developer.download.nvidia.com/presentations/2008/SIGGRAPH/HBAO_SIG08b.pdf (Accessed: 20 March 2024).

Bavoli, L. and Sainz, M.(2008) 'Screen Space Ambient Occlusion,' Available at https://www.researchgate.net/publication/228576448_Screen_Space_Ambient_Occlusion (Accessed: 20 March 2024).

Bavoil, L. and Sainz, M. (2009) 'Multi-layer dual-resolution screen-space ambient occlusion,' in *SIGGRAPH 2009: Talks*. New York, NY, USA: ACM, p. 45 (Accessed: 28 July 2024)

Chapman, J. (2013) 'SSAO Tutorial,' *John Chapman's Graphics Blog*, [blog] 2 January. Available at: https://john-chapman-graphics.blogspot.com/2013/01/ssao-tutorial.html (Accessed: 4 February 2024).

Cook, R.L. and Torrance, K.E., 1982. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1, pp.7–24. [Accessed 28 July 2024].

De Vries, J. (2020) 'LearnOpenGL,' *LearnOpenGL*, [online] Available at: https://learnopengl.com/Advanced-Lighting/SSAO (Accessed: 4 May 2024).

Filion, D. And McNaughton, R. (2008) 'Starcraft II Effects and Techniques', 'SIGGRAPH 2008' Available at https://www.yumpu.com/en/document/read/13581987/chapter05-filion-starcraftii (Accessed: 4 February 2024)

Landis, H., 2002. Production ready global illumination. In *SIGGRAPH 2002 Courses*, pp.331–338. [Accessed 28 July 2024].

Loos, B.J. and Sloan, P.-P., 2010. Volumetric obscurance. *Proceedings of the I3D 2010*. ACM, pp.151–156. Available at: https://www.ppsloan.org/publications/vo.pdf][Accessed 28 July 2024].

McGuire, M., Osman, B., Bukowski, M. and Hennessy, P., 2011. The Alchemy screen-space ambient obscurance algorithm. In *Proceedings of ACM SIGGRAPH / Eurographics High-Performance Graphics 2011 (HPG'11)*, High-Performance Graphics 2011, August. Vancouver, BC, Canada. Available at: https://casual-effects.com/research/McGuire2011AlchemyAO/index.html [Accessed 28 July 2024].

McGuire, M. (2010) 'Ambient Occlusion Volumes,' 'SI3D' [online] Available at https://www.researchgate.net/publication/220791988_Ambient_occlusion_volumes (Accessed: 4 February 2024).

McGuire, M. (2017) ‘Computer Graphics Archive), [online] Available at https://casual-effects.com/data/. (Accessed: 4 February 2024)

Meemknight, 2023. *hbao.frag* [Source code]. GitHub. Available at: https://github.com/meemknight/gl3d/blob/master/src/shaders/hbao/hbao.frag [Accessed 28 July 2024].

Mittring, M. (2007) ‘Finding Next Gen - Cry Engine 2’, Available at https://artis.inrialpes.fr/Membres/Olivier.Hoel/ssao/p97-mittring.pdf (Accessed: 28 July 2024)

Pharr, M. and Fernando, R. (2005) GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation (GPU Gems). Addison-Wesley Professional. (Accessed: 28 July 2024)

Spektre. (2022) Finding Point Inside Disk Defined by a Radius [Answer]. Stack Overflow. https://stackoverflow.com/questions/71965939/finding-point-inside-disk-defined-by-a-radius (Acessed: 28 July 2024)

Zhukov, S., Iones, A. and Kronin, G., 1998. An ambient light illumination model. In *Rendering Techniques ’98*. G. Drettakis and N. Max, eds., Eurographics, Springer-Verlag Wien New York, pp.45–56. (Accessed 28 July 2024)

Appendix A
Self-appraisal

A.1 Critical self-evaluation

As a whole, I would consider the implementation of the ambient occlusion methods as successful, however, there is still a lot of room for improvement. Not limited to but including the implementations themselves, the testing methodology, and the project management.

The methods used for implementing the ambient occlusion methods were based on papers or presentations done on them. These however were not very specific about implementation and thus a lot of variables could affect the performance of my implementations. All three could have been infinitely optimized in different ways, but I was unsure how I could fairly optimize them, as optimization techniques would differ between the two. This approach risked both increasing the workload and causing an unfair disparity in the performance of each method, essentially nullifying the validity of the comparison. In the end, however, a functional shader was produced for all three,

Using De Vries' (2020) SSAO implementation made the beginning process much smoother. It provided an outline of how my code should be set up and how shaders and framebuffers will interact with the application. De Vries' (2020) implementation of Starcraft II's SSAO is probably the implementation that best represents its respective method in this application. The HBAO used was based on Meemknight's (2023) implementation. The result produced was satisfactory but most likely not a completely accurate representation of NVIDIA's method. My implementation of Alchemy AO followed the paper's equations and calculation, as well as the sampling method but optimizations could have been made to improve it. In terms of visual quality, a more thorough Gaussian blur could have improved it. The disparity in implementation quality of each technique was probably the biggest fault in this project.

The method used to obtain results worked but could have been expanded on. Perhaps other models and scenes could have been used to further test each method. Although I do believe using FPS is a good numerical indicator of performance. On the other hand, it is much harder to judge the visual accuracy of each method. Analysis using my eyes leaves a lot of room for biases and inaccuracy and should be improved upon. One such improvement would be to implement ray-traced ambient occlusion which could act as a ground truth as it is highly accurate at simulating light, it is not used as a standard for real-time rendering due to performance costs but can still be used as a point of comparison. To take it further a full per-fragment brightness comparison could be done between the ray-traced scene and the other methods. This however was beyond my ability and the scale of this project.

A.2 Personal reflection and lessons learned

In reflecting on my dissertation journey, which centered around comparing Ambient Occlusion methods, I've garnered many insights and lessons that have significantly shaped my approach to both academic and practical aspects of computer graphics.

Planning and Time Management

One of the first lessons I learned was the importance of meticulous planning and effective time management. I was often stuck in certain phases of implementation, pushing me off my set timeline. This project reaffirmed that unexpected challenges often arise, requiring adjustments to timelines. Allowing for more leeway and being flexible with schedules is an important skill I learned and plan to better follow through with in the future.

Technical Skills and Problem-Solving: Delving into SSAO, HBAO and Alchemy AO enhanced my understanding of computer graphics and lighting. I faced numerous challenges, such as accounting for calculating using view space or world space or understanding the deferred shader rendering pipeline. Each problem required a unique solution, which improved my problem-solving skills and taught me the importance of a methodical approach to debugging and iterative development. I have become better at using the OpenGL API, C++, ImGUI and debugging tools like RenderDoc.

Research and Analysis: Conducting thorough research before beginning the practical implementation was crucial. It enabled me to build a solid foundation of knowledge about ambient occlusion techniques. Most papers lack simple explanations of these techniques so learning how to decipher their AO solutions was a rewarding challenge.

Persistence and Resilience: Perhaps the most personal lesson was about persistence. Facing repeated setbacks, especially with software bugs and performance issues, tested my resolve. Learning to persist through these challenges, maintaining a positive outlook, and adapting strategies as needed were crucial in seeing the project through to completion.

A.3 Legal, social, ethical, and professional issues

A.3.1 Legal issues

There should be no legal issues with the project as long as everything I use is under fair usage. Things such as the Crytex Sponza model need to be properly cited and referenced, as long as that is done correctly, there should be no legal issues.

A.3.2 Social issues

This may not necessarily apply to this paper but advancements in graphics are pushing the industry to release more graphically complex games and applications. Thus making it harder for those who may not have access to powerful devices to run these applications. This however is counteracted by the fact that the industry is also always pushing for more optimization and performance.

A.3.3 Ethical issues

A potential ethical issue of this project is its relation to graphics. Graphics drivers are becoming more powerful and are large consumers of energy. Given today's climate crisis, enhancements in this field may not be the best use of resources.

A.3.4 Professional issues

It is important that the outcome of this project is taken as a whole including the potential issues and inaccuracies of it. The results should not be looked at in isolation as someone implementing these methods could come to the wrong conclusions.

Appendix B

External Materials

Crytex Sponza Model

The Crytex Sponza Model was used in this project to test the ambient occlusion implementations. This is from the copyright.txt file given with the model.

“Sponza Model

http://www.crytek.com/cryengine/cryengine3/downloads

August 19, 2010

The Atrium Sponza Palace, Dubrovnik, is an elegant and improved model created by Frank Meinl. The original Sponza model was created by Marko Dabrovic in early 2002. Over the years, the Sponza Atrium scene has become one of the most popular 3D scenes for testing global illumination and radiosity due to its specific architectural structure which is particularly complex for global illumination light.

However, nowadays it is considered as a simple model, thus it was decided to create a new model with highly improved appearance and scene complexity. It is donated to the public for radiosity and is represented in several different formats (3ds, Obj) for use with various commercial 3D applications and renderers.

July 14, 2011, Morgan McGuire modified the model from Crytek's OBJ

export to correct some small errors. He computed bump maps from the normal maps using normal2bump.cpp (since MTL files expect height bumps, not normals), put the "mask" textures into the alpha channel of the associated diffuse texture, cleaned up noise in the masks, created the missing gi_flag.tga texture, and removed the long untextured banner floating in the middle of the atrium that appears in the file but in none of the published images of the model. The banner is in banner.obj.”

LearnOpenGL SSAO and Graphics Framework

This code was used as a base for the application (De Vries, 2020):

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/ssao.cpp

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/9.ssao_geometry.vs

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/9.ssao_geometry.fs

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/9.ssao.vs

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/9.ssao_lighting.fs

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/9.ssao.vs

https://learnopengl.com/code_viewer_gh.php?code=src/5.advanced_lighting/9.ssao/9.ssao_blur.fs

https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/shader.h

https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/camera.h

https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/model.h

https://learnopengl.com/code_viewer_gh.php?code=includes/learnopengl/mesh.h

HBAO Shader Code

The HBAO fragment shader is based on Meemknight's (2023) implementation of the code:

https://github.com/meemknight/gl3d/blob/master/src/shaders/hbao/hbao.frag

Diskpoint Function

The Diskpoint function is based on the following code from Spektre (2022).

https://stackoverflow.com/questions/71965939/finding-point-inside-disk-defined-by-a-radius