Democratic and Popular Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Mohamed Khider - BISKRA
Faculty of Exact Sciences, Natural Sciences and Life
Computer Science Department

Order Number: IVA1/M2/2018

REPORT

PRESENTED TO OBTAIN THE ACADEMIC MASTER DIPLOMA IN

COMPUTER SCIENCE

OPTION: ARTIFICIAL LIFE AND IMAGE

The Alchemy Screen Space Ambient Obscurance

By:
Rouabeh Younes

Defended the 25/06/2018, in front of the jury composed of:

Hamida Ammar	MAA	President
Zerari Abd Elmouméne	MAA	Supervisor
Benchabane Moufida	MAA	Examiner

University Year: 2017/2018

Dedication

i must express my very profound gratitude to my parents, my brothers and sisters, my best friends *for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching, writing and achieving this humble project.*
This accomplishment would not have been possible without them. Thank you. . .

Acknowledgements

First of all i want to thank Allah for giving me the courage and commitment to accomplish this work and for always guiding me to right path.

*Secondly i would like to express my sincere gratitude to my advisor Mr.**ABD EL MOUMÉNE ZERARI** for always being their for me when the curiosity overcomes, when i ran into trouble spot or had question about my research or writing, for his priceless advices that i had along this project or even before, for correcting the various faults that i've done or the misunderstanding that i had for several concepts.*

*I also want to express my thanks to Mr.**Hamida Ammar** for presiding the jury and also thank Mrs.**Benchabane Moufida** for the effort and time they took to read and evaluate this work.*

I also want to thank all computer science teachers of Mohamed Kheidher Unversity for their valuable lessons and advices during my study time.

I thank my fellow mates *for the great years that we spent together and all the fun we had.*

Abstract

Achieving a global illumination in real time has always been the interest of computer graphics where the need for approximations for rendering realistic scenes taking into account all kinds of interactions between light sources and objects of the scene or objects between themselves. We are interested in one of the methods used to approximate the global illumination which is the ambient occlusion that generates soft shadows in places with low light contribution to improve the realism of the scene. Although it has excellent results for approximating global illumination, it is not practical to use when real time is required because it depends on geometry in its calculations.

Alchemy Ambient Obscurance Method on the other hand is considered to be one of several solutions for approximating ambient obscurance which is an extended version of ambient occlusion using screen space. This method has effective results in real time, it causes a low performance, gives noisy results without speaking about ignorance of the important contributions of the scene.

The goal of this project is to implement the original Alchemy Ambient Obscurance Technique, to improve performance by using different filtering techniques rather than the original and to improve the samples by performing a comparative study between the different techniques of sampling to produce realistic and acceptable results.

Keywords: global illumination, ambient obscurance, ambient occlusion, soft shadows, screen space, filtering, sampling, real-time,

Résumé

Atteindre une illumination globale en temps réel a toujours été l'intérêt de l'infographie où la nécessité des approximations pour un rendu des scènes réalistes en tenant compte toutes sortes d'interactions entre les sources de lumière et les objets de la scène ou les objets entre eux-mêmes. Nous nous intéressons à l'une des méthodes utilisées pour approximer l'illumination globale qui est l'occultation ambiante qui génère des ombres douces dans des endroits à faible contribution lumineuse afin d'améliorer le réalisme de la scène. Bien qu'elle ait d'excellents rêsultats pour l'approximation de l'illumination globale, elle n'est pas pratique à utiliser lorsque le temps réel est demandé en raison du fait qu'il dépend de la géométrie dans ses calculs.

La méthode d'Alchemy obscurance ambiante d'autre part est considérée comme l'une des différentes solutions pour approximer l'obscurance ambiante qui est une version étendue de l'occultation ambiante en utilisant l'espace écran. Cette méthode a des résultats efficaces en temps réel, elle provoque une faible performance, donne des résultats bruyants sans parler sur l'ignorance des contributions importantes de la scène.

Le but de ce projet est d'implémenter la technique originale d'Alchemy obscurance ambiante, d'améliorer les performances en utilisant différentes techniques de filtrage plutôt que l'originale et d'améliorer les échantillons en effectuant une étude comparative entre les différentes techniques d'échantillonnage pour produire des résultats réalistes et acceptables.

mots clés: illumination globale, obscurance ambiante , occlusion ambiante , ombres douces, espace écran, filtrage, échantillonnage, temps-réel,

Contents

List of Figures

List of Tables

List of Abbreviations

SSAO	**S**creen **S**pace **A**mbient **O**cclusion
GPU	**G**raphics **P**rocessing **U**nit
GL	**G**raphics **L**ibrary
L	Luminance
E	Energetic
A	Area
BRDF	**B**idirectional **R**eflectance **D**istribution **F**unction
PDF	**P**robability **D**ensity **F**unction
RMS	**R**oot **M**ean **S**quared
VPL	**V**irtual **P**oint **L**ight
ISM	**I**mperfect **S**hadow **M**aps
AO	**A**mbient **O**cclusion
V	**V**isibility
HBAO	**H**orizon **B**ased **A**mbient **O**cclusion
VO	**V**olumetric **O**bscurance
CPU	**C**entral **P**rocessing **U**nit
R,G,B	**R**ed,**G**reen, **B**lue

General Introduction

From the beginning of computer graphics the virtual vision was one of its biggest interest in order to find a way to present reality or imagination graphically and to facilitate human beings life, where it all started by trying to figure out what is going on in the real life when it comes to the visualization mechanism resulting the essential component which is the light that is considered as the essence of our visual perception that is effected either by direct lighting, or indirect lighting or both of them together to result the realistic visualization mechanism which known as the *global illumination*.

Due the fact that the global illumination takes into consideration all types of interactions whether these types of interactions are direct or indirect. Accomplishing a full global illumination calculation in real-time is nearly impossible, and also demands powerful computers which are a lot expensive, and not possessed by all mankind that's why it is approximated rather than being exactly calculated.

One of the techniques that are used to approximate the global illumination and more specifically the indirect interactions is the *ambient occlusion*, which is defined as the amount of obstruction there is between a scene geometry and a light source, where each object is lit by taking all geometry into account resulting soft shadows in places like corners, cracks or even human body wrinkles, which add a quite amount of realism due the appearance of different geometry details by using the geometry itself which is considered to be a time consuming, and therefore it is not quite practical when in comes to a real-time need.

Several methods are used in order to achieve a scale down version of the ambient occlusion, using screen space rather than the geometry itself to enhance performance in real-time known as *screen space ambient occlusion (SSAO)*, by using pixel depth that is obtained from a depth buffer where the *AlchemySSAO* which is a SSAO method considered to use these depth informations to calculate a vector between the occludee and the occluder, which is used along other tuning parameters to approximate the ambient occlusion factor where the results are noisy due the usage of random sampling, and also low when it comes to performance which is caused by the chosen filtering method.

The objective of this project is to enhance the realism of a virtual rendered scene with highest performance, using the AlchemySSAO technique with a comparative study between the original applied sampling and filtering methods, and other suggested methods while using the power of GPUs to accelerate calculations, due the fact of its huge amount of cores that makes real-time rendering possible.

This project has been splitted into four chapters in order to accomplish the objective:

A background chapter as the first chapter, where we present the types of illuminations and going deeper to the foundation of the global illumination, along with its different fundamentals continuing with the essence of its approximation, which is the monte Carlo integration also going further to the global illumination approximation techniques.

In the second chapter we introduce the ambient occlusion, along with the ambient obscurance and the different ambient occlusion methods, which are followed by the screen space ambient occlusion and the different filtering techniques.

The third chapter contains the components that make this project possible, which are the GPUs and OpenGL where we go through the GPU architecture, and its different programming languages along the OpenGL evolution, and its different profiles and structures.

The implementation and results are defined as the last chapter, where we present the description and the different objectives of this project, along with the general design of the graphical application continuing with the application realization steps, and the different obtained results.

And lastly, we finish up with a general conclusion that contains the different accomplishments.

Chapter 1

Background

1.1 Introduction

From the first appearance of the computer graphics realistic real-time scene rendering has always been the interest of researchers while achieving this goal requires the study of a lot of phenomenons like light and materials nature, the interaction between them where everything is transformed into mathematical formulas that are used in order to achieve the closest realistic scene rendering with small amount of sacrifice that might be in time or quality, or maybe even performance if it is necessary.

Where the light interactions have been divided into two major types direct interaction which is known as *Local Illumination* and indirect Interaction known as *Global Illumination*, each of these types has a different light source understanding which cause eventually a time consumption that leads to real-time aspect loosing when it comes to the global illumination, contrary local illumination which can be calculated in real-time.

In this chapter, we start with the illumination different models, after that we go through the rendering equation of the global illumination, its demonstration, finally its different solutions which are based on approximation techniques.

1.2 Illumination

As we all know lighting a certain place can't be done without having light transported into the scene interacted with the different materials types causing light energy getting weaker each time the light hits a surface leaving an amount of energy to be redirected in multiple directions.

We mentioned in Section 1.1 that the light interactions are divided into two major types :

- *Local Illumination:* Which takes in consideration only light to surface interactions[1].
- *Global Illumination:* Which also takes in consideration surface-to-surface interactions[1].

1.2.1 Local Illumination

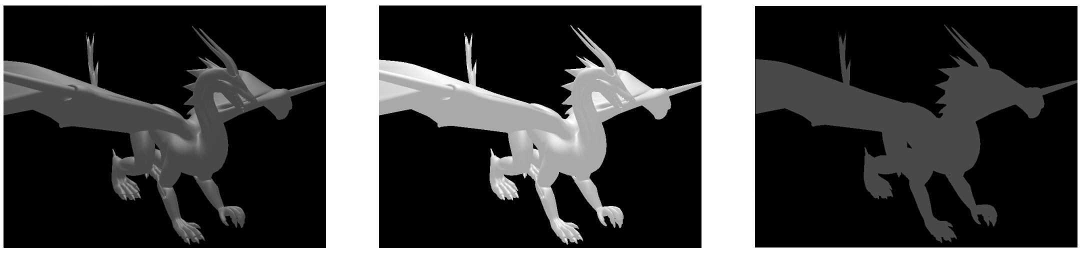
Illuminate scene elements locally needs a direct contact with the light source, which means that each element of the scene is lit as if there were no other elements in the scene and that's what exactly defines local illumination. Also scene elements are not all the same each element has its own material type which leads us to different types of reflections depending on how each surface react with the light source which is known as the illumination model, and of course a shading model to define in which way the illumination model is applied in order to achieve the element material[2].

1.2.1.1 Local Reflection Types

Local Reflections are resulted by an interaction between a light ray and an element of the scene as we mentioned in Section 1.2.1 where the element of the scene requires a material type taking for example *metal, mirroir* requires specularity, *walls* requires diffuse, *hair* requires diffuse and specularity, etc.... with a very simplified calculations that leads to a low degree of realism.

1.2.1.1.1 Ambient Reflection

Ambient reflection is used in most scenes to minimize unlit areas and to approximate global lighting by using a uniform value. Although it can be valuable, it should be used with caution such as in Figure 1.1(a) to avoid high brightness of all scene objects which leads to details vanishing like Figure 1.1(b), in fact uniform ambient light alone can't give details or even define an element shape such as in Figure 1.1(c) and that is not identical to the real world[2].



(A) Well controlled ambient light with specular, diffuse reflections

(B) Details vanishing caused by high ambient light value

(C) Absence of details caused by a uniform ambient light

FIGURE 1.1: *Ambient reflection component properties*

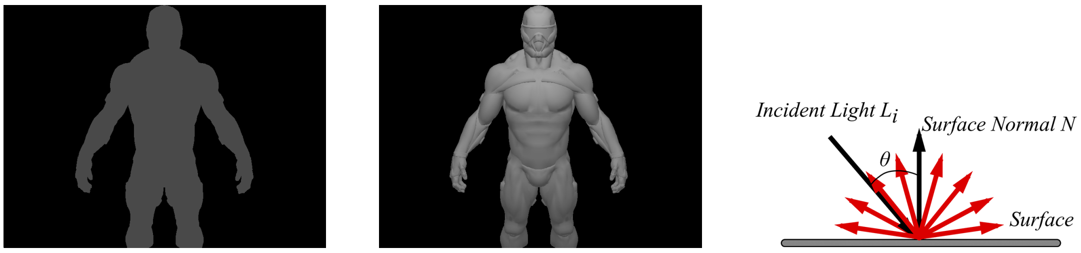
Ambient Reflection is calculated using the following formula :

$$I = I_a \cdot K_a$$

Where :
I_a is defined as low light constant intensity referred to as *Ambient Intensity*.
K_a is defined as ambient material of a certain object referred to as *Ambient Coefficient*.

1.2.1.1.2 Diffuse Reflection

Diffuse Reflection means that surfaces lighting depends on the visual light source and how each surface is facing it with a particular angle θ that can be calculated using a light source ray and a surface normal giving us the ability to calculate its cosine variable in a certain interval between $[0,1]$ where the 0 value means absolute darkness which leads to ambient lighting such as in Figure 1.2(a) while the 1 value means that the surface is full lighted, or partially lighted for the values between 0 and 1 like in Figure 1.2(b) which is known as *shading* leading to details appearance, where it doesn't matter which direction the eye is, but does matter which direction the light is as Figure 1.2(c) shows[3].



(A) Absence of diffuse reflection

(B) Details appearing due the diffuse reflection

(C) perfect diffuse reflection

FIGURE 1.2: *Diffuse reflection component properties*

Diffuse Reflection is calculated using the following formula :

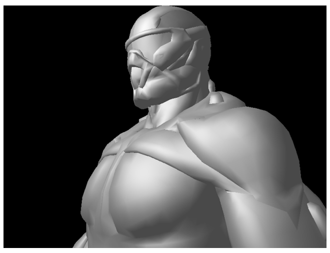
$$\begin{aligned} I &= I_d \cdot K_d \cos\theta \\ &= I_d \cdot K_d (\hat{N} \cdot \hat{L}) \end{aligned}$$

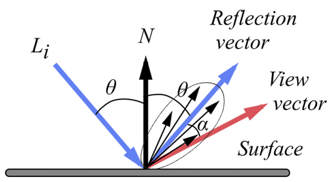
Where :
I_d is defined as light intensity referred to as *Diffuse Intensity*.
K_d is defined as Diffuse material of a certain object referred to as *Diffuse Coefficient*.
$\cos\theta = (\hat{N} \cdot \hat{L})$ is defined as the dot product between the normalized surface normal and a normalized light source ray.

1.2.1.1.3 Specular Reflection

Until now we only spoke about the ambient, diffuse reflection types that are needed usually a lot for matte surfaces like we mentioned in Section 1.2.1 as in Figure 1.3(a) while specular reflection is used in order to render a shiny object, or specular like metal, mirror using the view vector and reflecting a light source ray with the same angle taking the surface normal as a symmetry axis such as Figure 1.3(c) which is called the mirror reflection. In other words, real materials exhibit some amount of specular reflection Figure 1.3(b) around the mirror reflection angle where the very shiny surfaces specularly reflect over a very small range of angles. Less shiny materials reflect specular highlights over a wider range where the specular highlight is controlled using a specular power value[4].

(A) Absence of specular reflection matte surfaces


(B) specular high depending on eye view



(C) perfect specular reflection

FIGURE 1.3: *Specular reflection component properties*

Specular Reflection is calculated using the following formula :

$$I = I_s \cdot K_s \cos\alpha$$
$$= I_s \cdot K_s(\hat{R}\cdot\hat{V})^n$$

With : $\hat{R} = \hat{L} - 2(\hat{L}\cdot\hat{N})\hat{N}$ as the reflection vector. Which is resulted from :

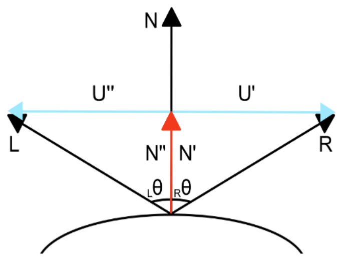


FIGURE 1.4: Reflection vector foundation

$$\theta_L = \theta_R$$
$$\hat{L}\cdot\hat{N} = \hat{R}\cdot\hat{N}$$
$$\hat{U}' = -\hat{U}"$$
$$\hat{U}" = \hat{L} - \hat{N}" = \hat{L} - (\hat{L}\cdot\hat{N})\hat{N}$$
$$\hat{U}' = \hat{R} - \hat{N}' = \hat{R} - (\hat{R}\cdot\hat{N})\hat{N}$$
$$\hat{R} - (\hat{L}\cdot\hat{N})\hat{N} = -(\hat{L} - (\hat{L}\cdot\hat{N})\hat{N})$$
$$\hat{R} - (\hat{R}\cdot\hat{N})\hat{N} = -(\hat{L} - (\hat{L}\cdot\hat{N})\hat{N})$$
$$\hat{R} = (\hat{L}\cdot\hat{N})\hat{N} - (\hat{L} - (\hat{L}\cdot\hat{N})\hat{N})$$
$$\hat{R} = (\hat{L}\cdot\hat{N})\hat{N} - \hat{L} + (\hat{L}\cdot\hat{N})\hat{N}$$
$$\hat{R} = 2(\hat{N}\cdot\hat{L})\hat{N} - \hat{L}$$

Where :
I_s is defined as the highest light intensity referred to as *Specular Intensity*.
K_s is defined as Specular material of a certain object referred to as *Specular Coefficient*.
$\cos\alpha$ = *($\hat{R}\cdot\hat{V}$)* is defined as the dot product between the normalized view vector and the normalized reflection vector.
n is defined as the material specularity degree referred to as *specular power*.

1.2.1.2 Local Illumination Techniques

Local illumination techniques depend only on a visual light source as we mentioned in Section 1.2.1, which means direct interaction of each object surface with a light source resulting different illuminations that is based on the used illumination model, where some of them are discussed the next few sections.

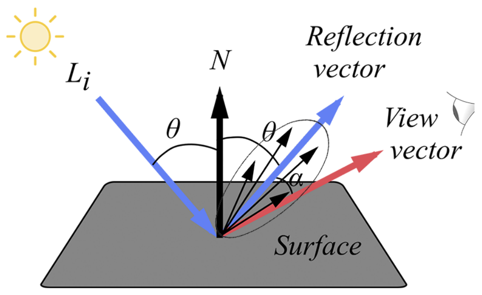
1.2.1.2.1 Phong Illumination Model

The Phong illumination model consists of being one of the first illumination models Figure 1.5(a) that has been introduced to the computer graphics in 1975[5], where the phong model consist of calculating the illumination at certain point using the three local illumination types mentioned in Section 1.2.1.1 which are the *Ambient Component*, *Diffuse Component* known as the ideal diffuse model *lambertian* and the *Specular Component*. All these three components combined allows us to calculate the *light intensity* at a certain surface point that depends on the observer location where the *light intensity* is emitted at the observer location as in Figure 1.5(b) and received constantly if the observer doesn't change its location or variably with each change of location[5].

Phong illumination model is calculated using the following formula :

$$I = I_a \cdot K_a + I_d \cdot K_d(\hat{N} \cdot \hat{L}) + I_s \cdot K_s(\hat{R} \cdot \hat{V})^n$$

(A) Buddha model using *phong illumination model*



(B) phong illumination model mathematical presentation

FIGURE 1.5: *Phong illumination model*

1.2.1.2.2 Phong Blinn Illumination Model

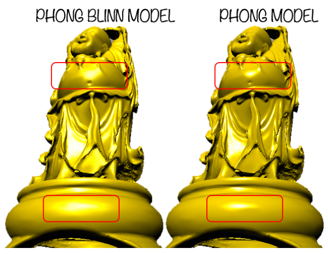
As we know the phong illumination model uses the mirror reflection vector that needs calculation for each pixel which is very expensive from performance side and gives unrealistic results see Figure 1.6(a). A halfway vector has been introduced in order to accelerate the calculation of the specular term by replacing the mirror reflection vector with the halfway vector Figure 1.6(c) producing a new enhanced model known as *phong Blinn*[6], which gives a better rendering results than phong in terms of realism as Figure 1.6(b) shows.

Phong illumination model is calculated using the following formula :

$$I = I_a \cdot K_a + I_d \cdot K_d(\hat{N} \cdot \hat{L}) + I_s \cdot K_s(\hat{H} \cdot \hat{N})^n$$

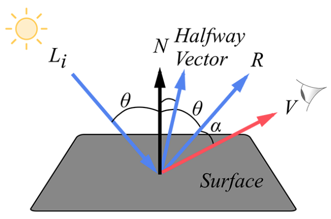
With The Halfway Vector equals to :

$$\hat{H} = \frac{(L+V)}{|L+V|} = (\hat{L} + \hat{V})$$



(A) Difference between *Phong* and *Phong Blinn* highlight

(B) buddha model using *Phong Blinn illumination model*



(C) Phong Blinn illumination model mathematical presentation

FIGURE 1.6: *Phong Blinn illumination model*

1.2.2 Global Illumination

Rendering realistic scenes always takes into consideration all geometry into account as in Figure 1.7 and in order to do those surface-to-surface interactions must be taken in consideration alongside with light-to-surface interactions, where these two types of interactions define the meaning of the global illumination as mentioned in Section 1.2 that takes into account the light energy distribution inside a scene resulting a high quality and realistic rendered images Figure 1.8(a) which obviously requires an amount of time conversely the local illumination where the scenes look more artificial Figure 1.8(b) than realistic[7].

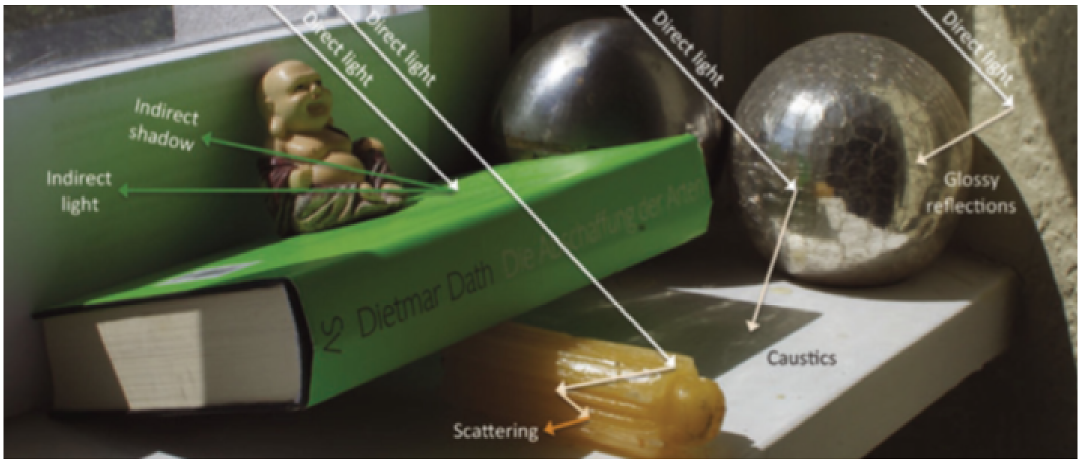
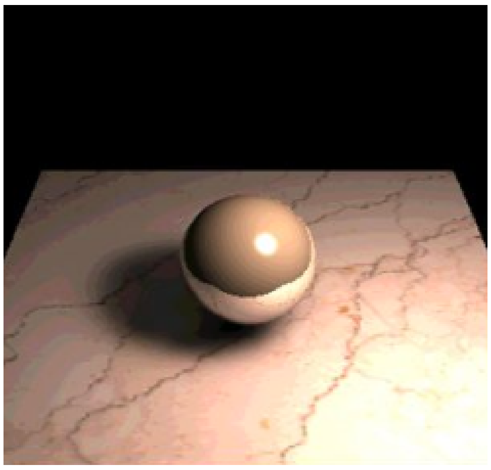
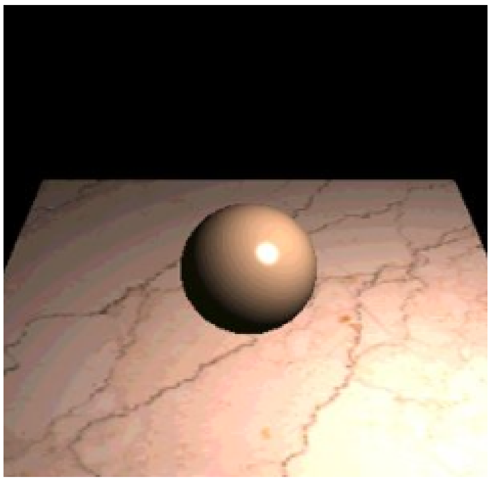


FIGURE 1.7: Global Illumination aspects demonstration using real-world photograph. Source: [8]



(A) Illumination at a point can depend on any other point in the scene



(B) Illumination depends on local objects and light sources only

FIGURE 1.8: Difference between local and global illumination. Source: [9]

1.3 Rendering Equation

A lot of aspects are needed in order to achieve the global illumination rendering equation which is complicated and needs to be resolved, where these aspects which are the basics are defined in the next few parts.

1.3.1 Solid Angle

The solid angle is the extension to 3D of the *standard 2D angle* see Figure 1.9(a). Instead of working on the unit circle in two dimensions, the projection of a given object into the unit sphere is calculated Figure 1.9(b).

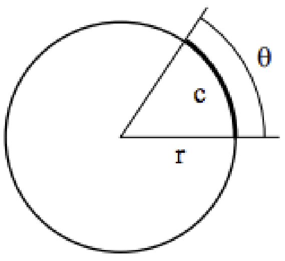
A solid angle is expressed in steradians(sr) where it is used for small areas, each point of the unit sphere has a spherical coordinates which are (r, θ, φ) as shown in Figure 1.9(c)[2].

The area of the surface dA at this point is given by:

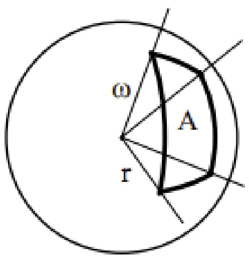
$$dA = r^2 \sin\theta d\theta d\varphi$$

The differential solid angle, $d\omega$, is given by:

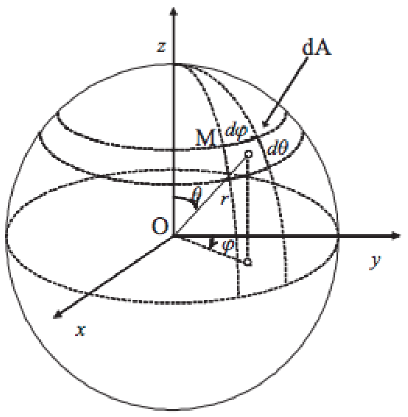
$$d\omega = \frac{dA}{r^2} = \sin\theta d\theta d\varphi$$



(A) 2-D angle



(B) 3-D angle



(C) *Solid Angle and Spherical Coordinates*

FIGURE 1.9: *Solid Angle* difference from 2-D space to 3-D space. Source: [10]

1.3.2 Radiance

Radiance L is defined as the radiant flux, or the quantity of energy Φ that is emitted in a given direction per unit solid angle $d\omega$ per unit area dA Figure 1.10 [2], which cause other surfaces illumination or a scene visualization.

Radiance L is defined by the following formula :

$$L(x \to \theta) = \frac{d^2\Phi}{dA^{\perp}d\omega} = \frac{d^2\Phi}{dA d\omega \cos\theta}$$

Where the flux radiant Φ is giving by :

$$\Phi = \frac{dQ}{dt}$$

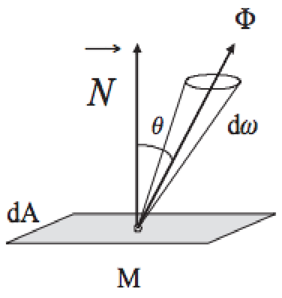


FIGURE 1.10: Radiance involved geometry. Source: [2]

1.3.3 Irradiance

While the radiance or luminance is the radiant quantity emitted in a given direction the *irradiance E* is completely the opposite, which means that it is the radiant quantity received by an object surface that comes from a light source known as *radiant flux* as Figure 1.11(a) shows, or comes from other surfaces known as the *radiant exitance* as in fig. 1.11(b)[10].

Irradiance mathematical formula demonstration:

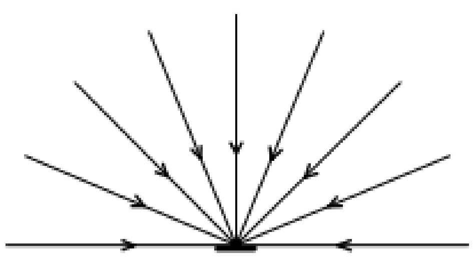
$$L = \frac{d^2\Phi}{dA^{\perp}d\omega}$$

$$= \frac{d\frac{d\Phi}{dA}}{d\omega\cos\theta}$$

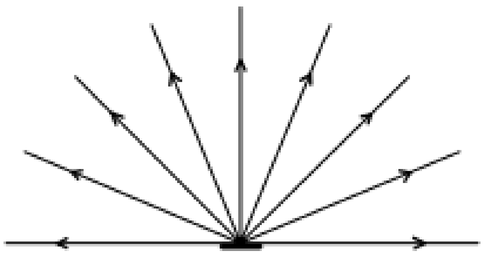
$$= \frac{dE}{d\omega\cos\theta}$$

$$\to dE = L d\omega \cos\theta$$

$$dE(x \leftarrow \psi) = L_i(x \leftarrow \psi)\cos\theta d\omega_{\psi}$$



(A) Irradiance "radiant flux"



(B) Irradiance exitance "radiant exitance"

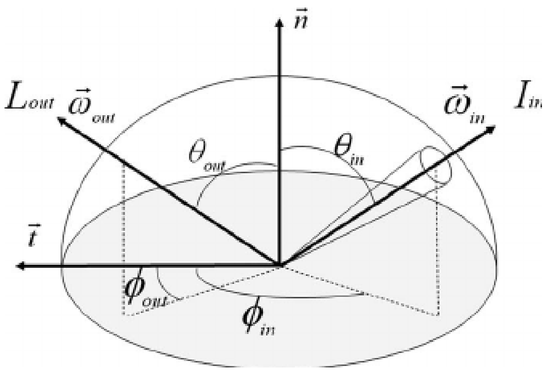
FIGURE 1.11: Different types of radiant receiving from a differential area. Source: [10]

1.3.4 Bidirectional Reflectance Distribution Function

In Section 1.2.1.2, we spoke about some of the simplified illumination models that don't require complications like the *BRDF* does Figure 1.12(a) and uses just a simple dot product operation to estimate the shading of a certain shape, and assuming that every surface is smooth surface which results unrealistic rendered images. Here the *BRDF* comes in handy which describes how an incident light is reflected by a rough surface point toward the viewer direction inside a hemisphere while respecting the energy conservation law which means that there is no loss in energy, everything is taking into account which gives a more realistic results like Figure 1.12(b) shows.

The *BRDF* is defined as follow :

$$fr(x, \psi \leftrightarrow \theta) = \frac{dL(x \rightarrow \theta)}{dE(x \leftarrow \psi)} = \frac{dL(x \rightarrow \theta)}{L_i(x \leftarrow \psi)\cos\theta d\omega_\psi}$$



(A) Geometry for the definition of the brdf. Source: [11]

(B) Dragon rendered with *MetalMaterial*, based on realistic measured gold scattering data. Source: [12]

FIGURE 1.12: Bidirectional Reflectance Distribution Function results and used Geometry

1.3.5 Rendering Equation Foundation

The rendering equation which makes rendering realistic scenes possible Figure 1.13 that was founded by Kajiya in 1986[13] is defined by taking in consideration all the real-world different light interactions or phenomenons. In other words as seen in Figure 1.7 in Section 1.2.2 where these interactions have no analytic solution when in comes to the *illuminace* part because of the massive relationship between each element of the scene that has been known to be one of the biggest research problems in the computer graphics history. In this Section 1.3.5 rendering equation is demonstrated step by step and some of the approximated solutions are presented in the following Section 1.3.6 with what is known as the *Monte Carlo Integration* in Section 1.4 that makes achieving an approximated global illumination almost in real-time possible.

In order to demonstrate the foundation of the *Rendering Equation*, we start with the part that has a sight of radiance (luminace) and irradiance (illuminace) part which is the *BRDF* formula that is mentioned in Section 1.3.4.

Redering Equation is given by the following formula:

$$L(x \to \theta) = L_e(x \to \theta) + L_r(x \to \theta)$$

Where:
$L_e(x \to \theta)$: is defined as the emitted spectrale radiance.
$L_r(x \to \theta)$: is defined as the reflected spectrale radiance.

The $L_e(x \to \theta)$ side of the equation which is the emitted spectrale radiance is easy part to calculate considering it as a one of the local illumination models cited in Section 1.2.1 which is a direct illumination as explained in Section 1.2 while the problem reside with the $L_r(x \to \theta)$ part that is the indirect illumination which comes from other elements of the scene toward a certain point.

From Section 1.3.4 we have:

$$fr(x, \psi \leftrightarrow \theta) = \frac{dL(x \to \theta)}{dE(x \leftarrow \psi)} = \frac{dL(x \to \theta)}{L_i(x \leftarrow \psi) cos\theta d\omega_\psi}$$

While $L_i(x \leftarrow \psi) cos\theta d\omega_\psi$ is considered to be the irradiance (Illuminace) as mentioned in Section 1.3.3 of a certain point.
Radiance (Luminance) is equal to:

$$dL(x \to \theta) = fr(x, \psi \leftrightarrow \theta).L_i(x \leftarrow \psi) cos\theta d\omega_\psi$$

With taking into account all the possible directions $d\theta$ the resulted formula is integrated on the hemisphere :

$$dL(x \to \theta) = \int_\Omega fr(x, \psi \leftrightarrow \theta).L_i(x \leftarrow \psi) cos\theta d\omega_\psi$$

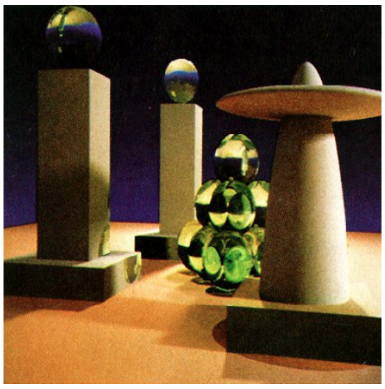


FIGURE 1.13: A sample image from the kajiya rendering equation result. Source: [13]

1.3.6 Global Illumination Techniques

In order to solve the rendering equation cited in Section 1.3, the reflected spectral radiance L_e integral part must be replaced with a summation which leads to precision loss caused by transferring from the continuous space $[x_1, x_2]$ to the discrete space $[s_1, s_2, s_3, ..., s_n]$ that allows a various of solutions either a non-interactive that are explained in this section 1.3.6 or an interactive solutions which we go through in the next Section 1.5.

1.3.6.1 Radiosity

The *radiosity* method consist of taking into account only ideal diffuse surfaces where it is also known as the radiant exitance see Figure 1.11(b) in Section 1.3.3, or in other words only Lambertian reflectors or emitters[2]. In Section 1.2.1.1.2 we come across that a *Lambertian* surfaces are independent of the viewing direction which leads to a great advantage for the radiosity method that is calculating the visual scene only once, conversely the *ray tracing* that needs to calculate the illumination each time the observer moves[14], even if radiosity is defined as a non-interactive method it gives a full realistic scene rendered images when it comes to the diffuse component as Figure 1.14 illustrate.

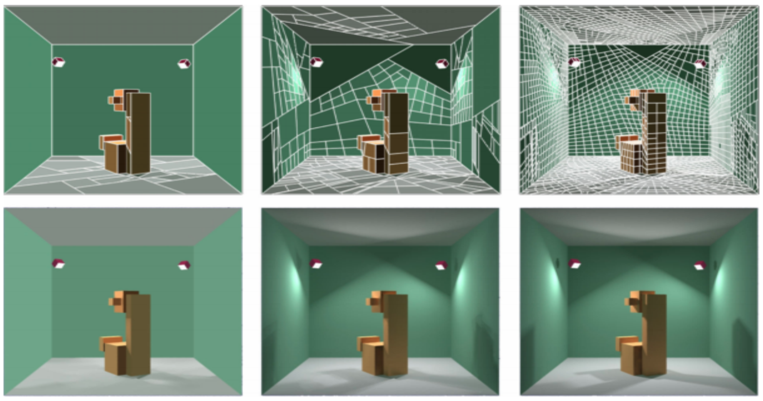
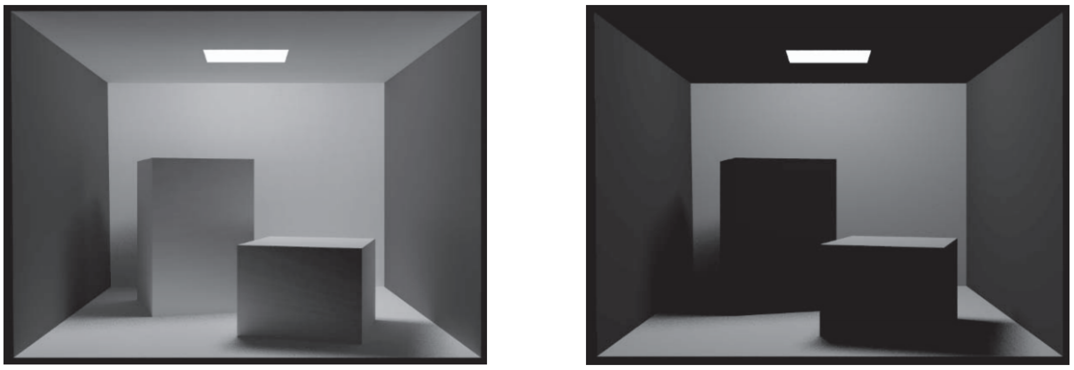


FIGURE 1.14: Radiosity approches discretize the scene into patches and compute the indirect illumination. Source: [15]

1.3.6.2 Path Tracing

In this section, we explain the *Path Tracing* Algorithm which is an extended ray tracing algorithm because of the absence of the exact diffuse component aspect that is caused by taking in consideration only one direction of illumination with the light source direction like Figure 1.15(b) that is defined as direct illumination, the path tracing comes to fill that gap by using the Monte Carlo sampling techniques which we go through in the next section to provide a global illumination calculation in a virtual scene, where the indirect lighting is calculated by tracing rays in all directions surrounding the point of intersection on the surface resulting a high level of realism as in Figure 1.15(b)[2].



(A) Path-traced Cornell room (B) Raytraced Cornell room

FIGURE 1.15: The power of Global Illumination in achieving realistic rendering. Source: [2]

1.4 Monte Carlo Integration

We mentioned in Section 1.3.5 that Rendering Equation has no analytic solution, which means that a higher dimensional and discontinuous integral needs to be solved in order to estimate the light distribution in a virtual scene which requires numerical integration techniques[12].

Although standard numerical integration methods either are deterministic such as *Gaussian quadrature* which are effective in solving a higher dimensional integral but convergence rate is poor which means that virtual scene rendering takes a quite amount of time and no expected results. On the other hand, stochastic methods as the *Monte Carlo Integration* are used during the fact that they depend on randomness for integrals evaluations with a convergence rate that is independent of the integrand dimensionality[12]. In this section, we review important concepts from probability and Monte Carlo techniques to evaluate the key integrals in rendering.

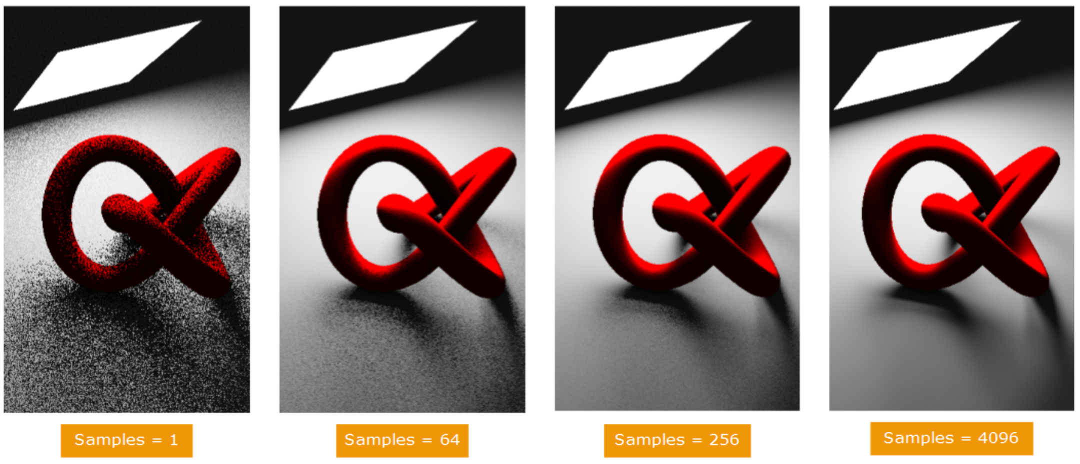


FIGURE 1.16: Monte Carlo sampling applied to shadow casting.
Source: [16]

While The *Monte Carlo Integration* is one of the most powerful and simplest used methods in evaluation rendering equation integrals. But it has an inconvenient which is the large variance of its samples that requires using a large set of samples [17] in order to achieve the wanted results as Figure 1.16 shows.

The monte carlo ingeration sampling large variance leads to an unwanted effect that is resulted in rendered images known as *Noise* due to the low number of used samples, which needs to be high in order to reduce the variance and converge into the wanted solution. A lot of techniques are developed to provide variance reduction as the *Importance Sampling* that target the finest or wanted solution, or using the randomness aspect to generate random values which contribute to reach the wanted solution[12].

1.4.1 Probabilites Theory Basics

The main essence of probability theory is an experiment that can be repeated, at least hypothetically, under essentially the same conditions and that may give different outcomes on each time we repeat the experiment [17].

Where the possible set outcomes of an experiment are called "sample space" Ω and each outcome of trial is known as a continuous random variable x, where its behavior is entirely described by the distribution of values it takes. This distribution can be quantitatively described by the *probability density function* with $x \sim p$[17].

The probability of x assumes a particular value in some interval $[a,b]$ is given by:

$$Probability(x \in [a,b]) = \int_a^b p(x)dx$$

The *pdf* has two characteristics:

$$p(x) \geq 0 \qquad (Probability\ is\ nonenegative)$$

$$\int_{-\infty}^{+\infty} p(x)dx = 1 \qquad (Probability(x \in \mathbb{R}) = 1$$

The average value that a real function f with a pdf takes is called *expected value* giving by :

$$E(f(x)) = \int_{\Omega} f(x)p(x)dx$$

The $E(x)$ of a random variable can be calculated by setting $f(x) = x$.

1.4.2 Monte Carlo Estimator

One of the most important things in the monte carlo integration is the *Monte Carlo Estimator* that allows any function f integral estimation by a summation of its values in a sampling points using the pdf value at the same sampling points[2].

The Monte Carlo Estimator is given by the following formula :

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}$$

Where:
N is the number of samples or "$kernelsamples$".
I is the the desired integral to estimate.
$\frac{f(x_i)}{p(x_i)}$ is the key part which makes the samples weighted that means higher samples contribution in regions with low density and less contribution in regions with high density.

Demonstration of the Monte Carlo Estimator formula:

$$E[\langle I \rangle] = E\left[\frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)}\right] = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{f(x_i)}{p(x_i)}\right] \approx \frac{1}{N} N \int \frac{f(x)}{p(x)} p(x)dx = \int f(x)dx = I$$

1.4.3 Convergence rates

The convergence of Monte Carlo integration is computed by using the variance of its estimator $\langle I \rangle$[18]. For simplicity Let $Y_i = f(X_i)/p(X_i)$, so that:

$$\langle I \rangle = \frac{1}{N} \sum_{i=1}^{N} Y_i$$

Also let $Y = Y_1$. We then have :

$$V[Y] = E[Y^2] - E[Y]^2 = \int_{\Omega} \frac{f^2(x)}{p(x)} p(x)dx - I^2$$

Now considering the quantity is finite, it won't be hard to discover that the variance of $\langle I \rangle$ is decreased linearly with N:

$$\sigma^2[\langle I \rangle] = \sigma^2\left[\frac{1}{N}\sum_{i=1}^{N} Y_i\right] = \frac{1}{N^2}\sigma^2\left[\sum_{i=1}^{N} Y_i\right] = \frac{1}{N^2}\sum_{i=1}^{N}\sigma^2[Y_i] = \frac{1}{N}\sigma^2[Y]$$

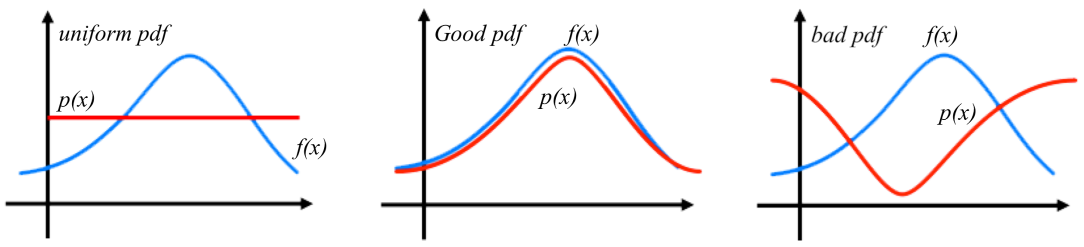
While the fact that the Y_i are independent samples. The $\sigma^2[aY] = a^2\sigma^2[Y]$, which gives the standard deviation :

$$\sigma[\langle I \rangle] = \frac{1}{\sqrt{N}}\sigma Y$$

Which leads us to consider that the RMS error converges at a rate of $O(N^{\frac{-1}{2}})$ that means in order to reduce RMS error by 2 the sampling number must be increased four times.

1.4.3.1 Importance Sampling

During to the variance slowest convergence as mentioned in the previous section, an *Importance sampling* technique is used in order to reduce the estimator variance and while each random value of the integrand f is divided by its *pdf* due to the weighting process as we explained in Section 1.4.2, a similar density function p to the integrand f must be chosen as in Figure 1.17(b)that makes the integrand function looks more constant according to its *pdf* where $p(x) = cf(x)$ is preferable to be the best choice[18].



(A) Uniform PDF due the uniform sampling which has the highest convergence rate

(B) Similar PDF or Good PDF in other words which gives the smallest convergence rate error

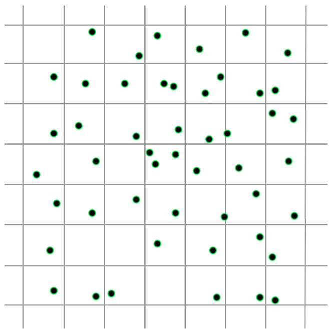
(C) Bad PDF which must be avoided and stick with the uniform PDF in cases of complications

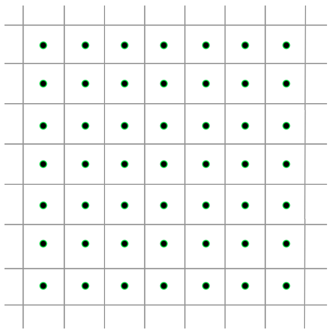
FIGURE 1.17: for *importance sampling* to work, you need a PDF (in red) which is as close as possible to the integrand function (in blue)

Importance Sampling considered to be the essence of the monte carlo integration, if a good pdf is chosen which is not practical and not always the case due to the absence of the desired integral value to compute the normalization constant c that leads to 0 variance[18]. By other meaning a full disappearing of the noise effect in rendered images rather than that we just stick with decreasing the variance value.

1.4.3.2 Distributions Techniques

There is another strategy that comes along variance reduction which is ensuring that samples are distributed more or less uniformly over the domain[18] that leads to a faster convergence. Conversely picking samples randomly that causes missing important areas as in Figure 4.4(a), while uniform sampling which is illustrated in Figure 4.4(b) solve this problem it causes the aliasing effect, the stratified technique Figure 4.4(c) comes to replace the aliasing effect with *noise* that leads to less perceptual human vision[19]


(A) randomly distributed samples over 17x17 grid


(B) uniformly distributed samples over 17x17 grid

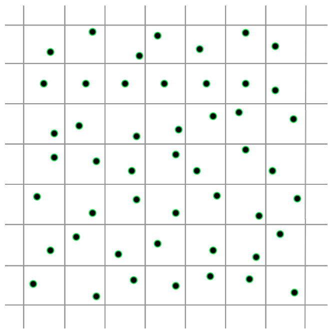

(C) Stratified distributed samples over 17x17 grid

FIGURE 1.18: Monte Carlo integration different distribution techniques using $17x17$ samples kernel

1.5 Global Illumination Approximation Techniques

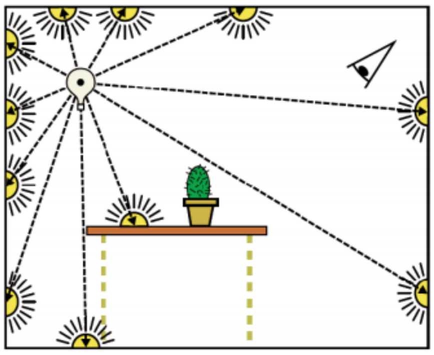
In the previous Section 1.3.6, we came across the global illumination techniques where it is well known that the used algorithms demand a quite amount of time in order to render a realistic images which is unusable in Real-Time applications like *Video games* for example, which requires the interactivity aspect and at the same time a realistic rendered images aspect where gathering these two under one roof is impossible without having the *approximation* term included. In this section, we cite the most known global illumination approximation techniques.

1.5.1 Instant Radiosity

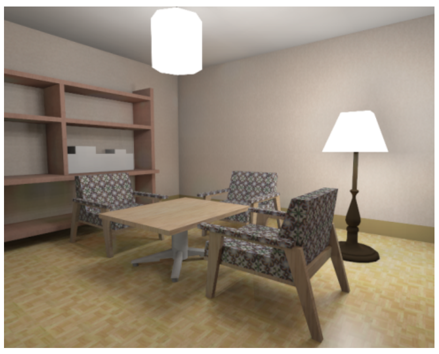
Instant Radiosity has been introduced to the computer graphics field by Keller[20], which consist of replacing the secondary light bounce by a virtual light points referred to as (*VPLs*) as Figure 1.19(a) shows.

Where the algorithm of the instant radiosity depends on the monte carlo integration techniques, and to be more specifically a quasi-monte carlo integration that is used in order to approximate the diffuse radiance in a scene by generating a particles approximation of the diffuse radiance[20].

After that the power of GPU is used in order to render shadows for each particle that has been become a virtual light source, finally summing up the rendered images that contains shadows produce a global illumination approximation Figure 1.19(b)[20].



(A) Traced Paths from the primary light. Source: [21]



(B) Instant Radiosity results. Source: [20]

FIGURE 1.19: Instant Radiosity Indirect Illumination approximation

1.5.2 Reflective Shadow Maps

Reflective Shadow Maps are defined as considering each pixel of shadow maps pixels as an indirect light source, which generates the one-bounce indirect illumination in a scene with a single light source where all the surfaces visible in its shadow maps are considered as one-bounce indirect illumination, what makes the shadow maps contains indirect illumination information, which leads to considering that the reflective shadow maps are collection of textures as Figure 1.23 shows that contains information of surfaces visibility from a light source where this collection is sampled to select the surfaces that are used as a virtual point lights like Figure 1.20 shows[22].

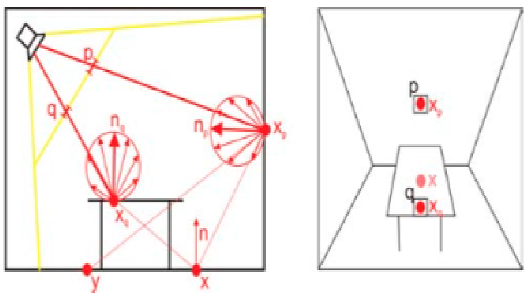


FIGURE 1.20: Two indirect pixel lights corresponding to two pixels. Source: [22]

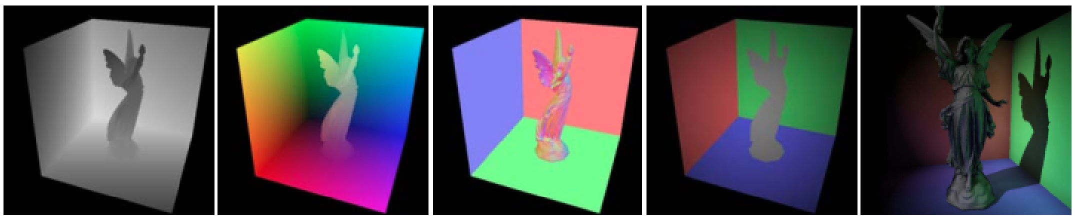


FIGURE 1.21: Reflective Shadow Maps components (depth, positions, normal, flux) and resulted imaged rendered using the Reflective Shadow Maps. Source: [22]

1.5.3 Imperfect Shadow Maps

Imperfect Shadow Maps Shadow Maps (ISM) fril the other hand that was introduced in 2008[23], uses low-resolution point representation in order to capture visibility in the scene rather than high-resolution shadow maps[24].

Where the Imperfect Shadow Maps uses a different scene geometry representation to make shadow maps rendering rapidly enhanced, by using geometry points as Figure 1.22 illustrate rather than using polygons which can provide the imperfect shadow maps for indirect illumination calculation[23], in order to render a large scale of shadow maps in just one draw call with less amount of geometry reduction.

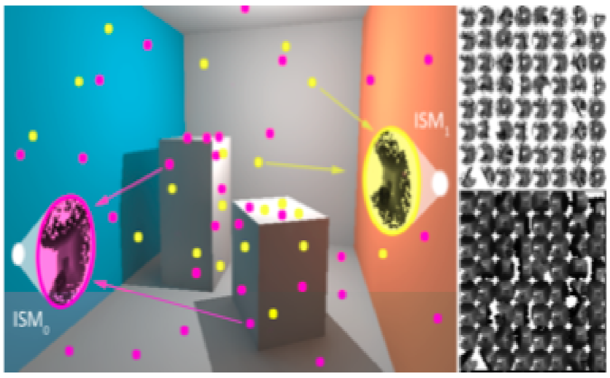


FIGURE 1.22: Global Illumination with imperfect shadow maps illustration. Source: [23]

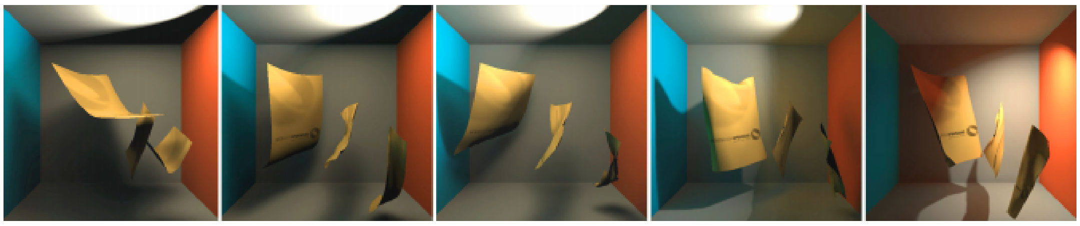


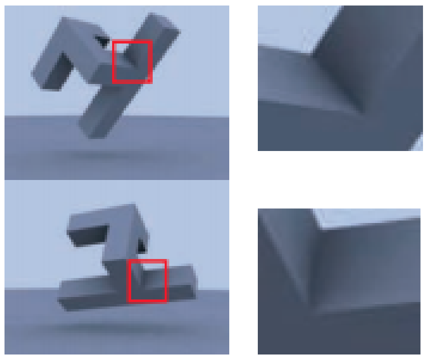
FIGURE 1.23: Indirect illumination of a dynamic scene using Imperfect Shadow Maps. Source: [23]

1.5.4 Ambient Occlusion

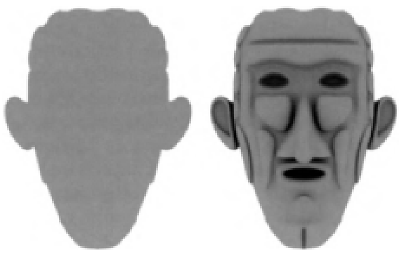
Ambient Occlusion is a technique that takes in consideration light attenuation due to occlusion. In another meaning, using AO makes scene parts occluded by nearby objects for example, corners or cracks Figure 1.24(a) or even human body wrinkles are more likely occluded than flat surfaces by making them receive less contribution from the shading model resulting soft shadowed areas. Ambient Occlusion is popular technique in interactive application specially *Videogames* for its ability to provide a substantial improvement in the shading realism and enhancing the scene visualization Figure 1.24(b) rather than just using a constant ambient light Figure 1.24(b) as mentioned in Section 1.2.1.1.1[25].

Where the ambient occlusion is considered to be a global illumination approximation method that depends only on geometry location, resulting one-dimensional value that is between $[0, 1]$ used to define how much each scene point is exposed to a to illumination process.

Regardless of the significant realism improvement by the AO its quite a time-consuming technique which is caused by the traced rays that defines how much a point as blocked by nearby objects[25].



(A) Ambient Occlusion realism enhancement in corners. Source: [26]



(B) Ambient Occlusion versus a constant ambient term. Source: PRT-Demo, Microsoft SDK, November 2007

FIGURE 1.24: Ambient Occlusion and its visual improvement in rendering

1.5.5 Comparison

In this chapter section, we came across a comparison between the global illumination approximation techniques mentioned in Section 1.5 by citing each technique advantages and disadvantages see Table 1.1.

Technique	Advantages	Disadvantages
Instant Radiosity	- Efficient for Diffuse surfaces, (can be extended to specular surfaces). - Provide much more realism interactively. - On the fly computed solution displaying. - Low memory requirements.	- Time consuming technique due the VPLs number. - View dependency. - Final result quality depends on GPU capabality.
Reflective Shadow Maps	- Geometric Bounce - Dynamic lighting	- Indirect light sources aren't occluded
Imperfect Shadow Maps	- High Performance in Real-Time	- Shadow Maps for higher resolution needs
Ambient occlusion	- Light Source independent technique. - Soft Shadows are rendered. - Highly improved scene realism.	- Geometry dependent technique, (Complex Scenes leads to time consumption). - Applicable just for Diffuse Environments.

TABLE 1.1: Global Illuminations Approximation Techniques

1.6 Conclusion

In this chapter, we came across the foundation of the global illumination and its various aspects which are included in its powerful equation that has been demonstrated by extracting the radiance from the brdf, which makes scene rendering realistic as much as it could either in deferred time using high-resolution techniques, or an approximation techniques that sacrifice a bit of realism in order to provide us the interactivity in real-time which is a very important aspect in computer graphics fields.

Where the Ambient Occlusion seems to be the fastest and the simplest global illumination approximation technique that makes enhancing scene realism more alive that before either statically, or dynamically.

Chapter 2

Screen Space Ambient Occlusion

2.1 Introduction

The ambient occlusion add a quite amount of realism enhancement to the rendered scenes by approximating the global illumination in places where the light intensity has a small or none contribution at all such as cracks, corners, and even the human body wrinkles by surrounding each blocked point or occluded in more accurate way by generating soft shadows which also makes objects shapes or "details" more visual than before.

Regardless the realism enhancement of scene rendering that is calculated using the ambient occlusion that provides a good ratio between quality and speed, but it was never practical to run it completely in real-time for complex geometry in applications such as video games, where it needs to be computed per frame within few milliseconds.

Screen Space Ambient Occlusion a technique that has achieved approximating real-time ambient occlusion computation by completely rely on images and discard all geometry dependency and complexity.

In this chapter, we start by the ambient occlusion foundation and its various methods, after that we come across the screen space and its powerful usage when it comes to the ambient occlusion computation with the most recent and used screen space ambient occlusion.

2.2 Ambient Occlusion

Global illumination approximation in most often used techniques requires knowledge of how geometry is positioned in a scene the ambient occlusion has been introduced in 2002[27] which approximates the amount of light reaching a point on a diffuse surface based on its directly visible occluders[28].

The essence behind the ambient occlusion is that it compute how much each surface point is exposed to light intensity which provide a quite approximation of how much each point is soft shadowed using traced rays around each surface occluded point.

A Cosine-weighted distribution is used between a surface normal and a traced ray in a hemisphere domain Ω as Figure 2.1 shows.

Where the ambient occlusion is demonstrated as follow:

We have the Rendering Equation equals to:

$$L(x \to \theta) = L_e(x \to \theta) + \int_\Omega fr(x, \psi \leftrightarrow \theta).L_i(x \leftarrow \psi) cos\theta d\omega_\psi$$

We assume that the scne uses only a diffuse model so that all the emmited light L_e is coming from another source (environment map for example). This gives us: $L_e(x \to \theta) = 0$. This removes the left term of the sum, so we have:

$$L(x \to \theta) = \int_\Omega fr(x, \psi \leftrightarrow \theta).L_i(x \leftarrow \psi) cos\theta d\omega_\psi$$

Now, let's choose a BRDF. For simplicity's reason, we use Lambert's model, so $fr(x, \psi \leftrightarrow \theta) = \frac{c}{\pi}$ with c the surface color, now we get:

$$L(x \to \theta) = \frac{c}{\pi} \int_\Omega L_i(x \leftarrow \psi) cos\theta d\omega_\psi$$

Because fr is a constant, it is outside the integral.

Now, we simplify the incoming radiance from any direction side L_i as a simple visibility function $V(\omega)$ which is for occluder existent or nonexistent given as follow:

$$V(\omega) = \begin{cases} 0, & \text{Ray from p in direction } \omega \text{ hits anything} \\ 1, & \text{Otherwise} \end{cases}$$

The unoccluded part of the hemisphere is not hard to compute but a stronger mathematical is demanded in order to evaluate the integral[29].

Also, we set $c = 1$, because we assume that the surface is fully reflective, so the value 1 won't affect the color, the equation becomes:

$$A0(p) = \frac{1}{\pi} \int_\Omega V(\omega)(n \cdot \omega) d\omega$$

The integral needs to be evaluated by considering the quantity is finite monte carlo integration is used to evaluation process which leads to the following formula[29]:

$$AO \approx \frac{1}{N} \sum_{k=1}^{N} V \cdot (n \cdot \omega_k)$$

Where:
AO is the Ambient Occlusion approximated factor.
N is the samples number in other words rays number. V is the previous discussed visibility function. $(n \cdot \omega_n)$: is the key part which is the dot product between a ray sample ω_n and the surface normal n.

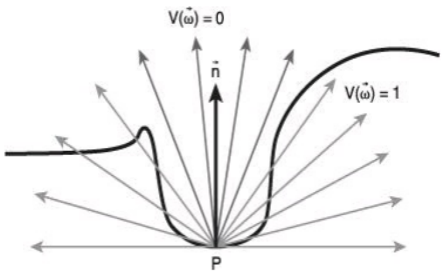


FIGURE 2.1: Rays distributed in the normal-oriented hemisphere. Source: [28]

2.3 Ambient Obscurance

Ambient Occlusion gives a quite good approximation of occluded surfaces indirect lighting which makes the rendered scene looks more realistic than before, but using Ray Marching methods in almost closed environments makes results much more darker that leads even to details disappearing because of the small amount of rays that could escape the scene which means that the visibility function of the AO V is in the most cases equals to 0 leading to make scene getting darker until reaching fully blacked scene as Figure 2.2 illustrates

Ambient Obscurance was introduced in 1998[30] which makes AO no longer depends on the visibility function V but to be in more accurate way by using an attenuation function usually called *falloff function* $\rho(d)$ that use the distance d into account.

By taking the distance between a point and an occluder into account occluders became having a little or no influence at all while calculating the ambient occlusion factor which that the rendered scene is more brighter than using the visibility function[29].

The falloff function has the following properties:

ρ is monotonically increasing function of d.

The distance is considered finite by setting $\rho(d) = 1$ when $d > d_{max}$.

The Ambient Obscurance formula factor is given by the following formula:

$$AO^*(p) = \frac{1}{\pi}\int_{\Omega}\rho(|p_\omega - p|)(n\cdot\omega)d\omega$$

Where:
$|p_\omega - p|$ is defined as the distance between a point p and the ray ω first hit.

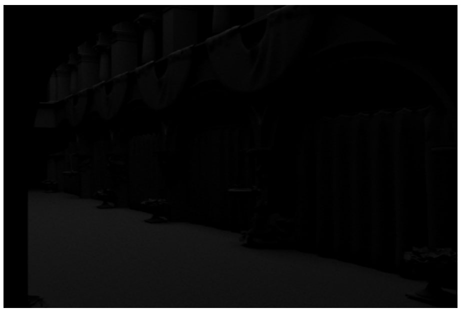


FIGURE 2.2: Ambient Occlusion in enclosed scenes results Source: [29]

2.4 Real-Time Ambient Occlusion Methods

When it comes to the ambient occlusion computation a lot of methods has been provided to the computer graphics fields where each of these methods compete to gain real-time aspect interactivity by depending on various types of computation:

2.4.1 Object Based Methods

Geometry based methods depends on the geometry itself to calculate the ambient occlusion factor where these kind of methods don't rely on traced rays, so they are more efficient than ray-traced ambient occlusion which leads to real-time interactivity achievement as we present some of these methods in the next part[31].

2.4.1.1 Ambient Occlusion Fields

Ambient Occlusion fields technique consist of an inter-object ambient occlusion calculation, where each occluded object has a pre-computation field in its surrounding space that leads to an approximation of the occlusion that has been caused by a scene object. where each field which is a volumetric information is then used for shadow casting on the objects that have the occlusion influenceFigure 2.4[32].

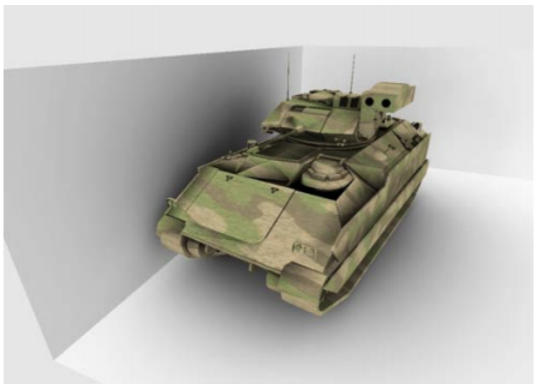


FIGURE 2.3: Ambient Occlusion Fields results.. Source: [32]

2.4.1.2 Fast Precomputed Ambient Occlusion for Shadows Proximity

Fast Precomputed ambient occlusion for shadows proximity considered as a new method for ambient occlusion evaluation or more accurate stocking which is an efficient and simple that uses just a small amount of graphic memory in order to store and retrieve the ambient occlusion values which has no relationship with the occluder complexity which makes this method recommended for Real-Time needs[33].

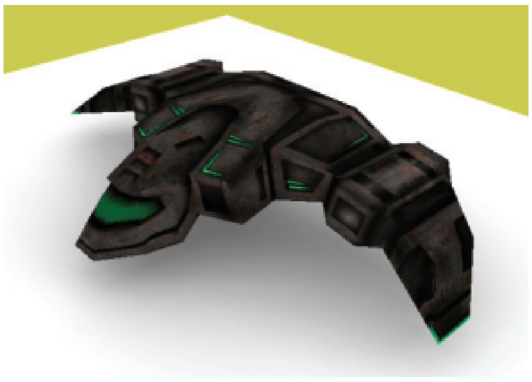


FIGURE 2.4: Contact Shadows illustration. Source: [33]

2.4.2 Point Based Methods

Point based methods consider ambient occlusion factor computation as a process that is done in each object point rather than the whole geometry itself where we present some of these methods in the next part.

2.4.2.1 Dynamic Ambient Occlusion and Indirect Lighting

Dynamic Ambient Occlusion and Indirect is a technique that has been introduced by Bunnell[34] of Nvidia that consist of using the gpu for computation acceleration of each Ambient Occlusion and Indirect Lighting that turns algorithms usage in a real-time possible which can be quite useful for dynamic.

Where the polygonal data in converted to surface elements which makes illuminated or shadowed surfaces by a part of a surface calculation easy, each surface element is defined as an oriented disk with the geometry informations which are the position, normal, area as Figure 2.5 shows. An element has a front face and a back face that makes the front face responsible for light emitting and reflecting and the back face has the shadows casting and light transmitting where for each object point an element is created[34] which result a quite amount of realism illustrated in Figure 2.7.

2.4.2.2 Hardware Accelerated Ambient Occlusion Computation

Hardware Accelerated Ambient Occlusion Computation uses the gpu to accelerate the visibility between surface points and directional light sources computation[35].

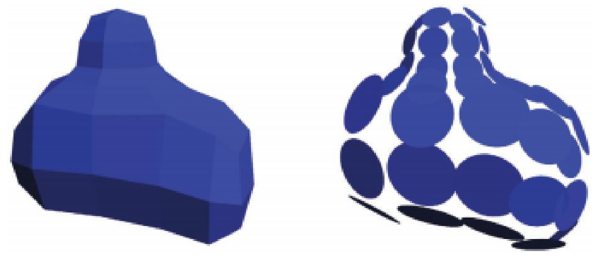


FIGURE 2.5: Polygonal data converting to disk-shaped elements. Source: [34]

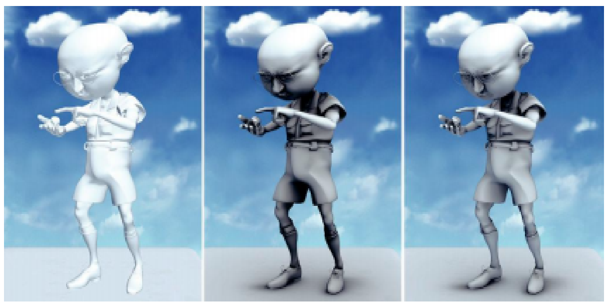


FIGURE 2.6: Realism enhancement with ambient occlusion and indirect lighting computation. Source: [34]

Which gives an approximation of the rendering equation using gpu by accumulating depth tests of each vertex fragment as it is captured from a light source direction, where this methods doesn't require a preprocessing stage of scene object and no memory lost costs that can be used either with static or dynamic geometry and can handle a large polygonal models[35].

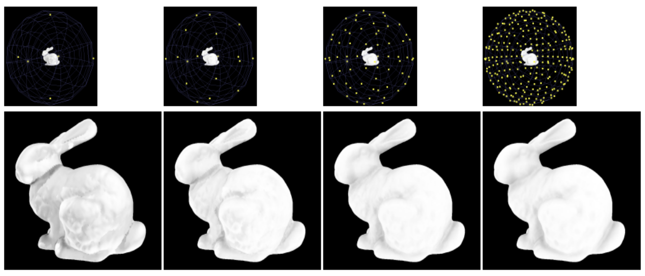


FIGURE 2.7: Standford Rabbit rendered with different light sources directions. Source: [35]

2.4.3 Screen Space Based Methods

We mentioned in the previous sections the ambient occlusion computation methods that are based on geometry now when it comes using images as a source for calculating the ambient occlusion screen space methods are the finest ones where the computation depends completely on images rather than geometry which is known that it takes a quite amount of time below we present most of the screen space used methods.

2.4.3.1 Screen Space Ambient Occlusion

Screen Space Ambient Occlusion is a technique that has been introduced by Crytek in 2007[36] by its creative developer Vladimir Kajalin and used for the first time an efficient way for ambient occlusion calculation in a video game called *Crysis* which has gained a lot of popularity in that time due the fast and simplest ambient occlusion approximating computation, where this technique consider the depth buffer as a geometry state approximation and as a pillar for the AO computation at each rendered pixel by sampling its surrounding pixels and verifying if a sample pixel is above or below the center of the sampling that is considered as the point we want to calculate the AO at which gives a quite good approximation of the ambient occlusion factor.

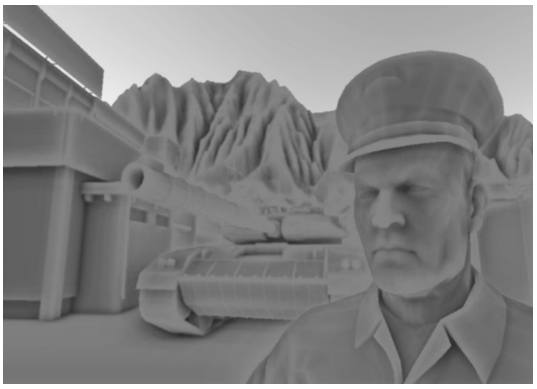
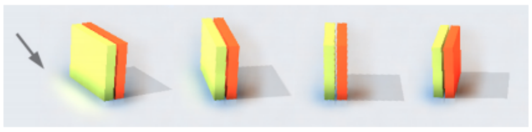


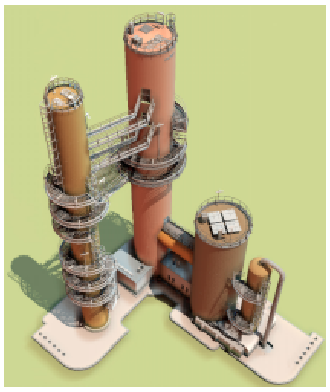
FIGURE 2.8: Ambient Occlusion Approximation using Screen Space.. Source: [36]

2.4.3.2 Screen Space Directional Occlusion

Screen Space Directional Occlusion allows calculate local indirect illumination and cast directional like in Figure 2.9(a) without the aid of environmental maps which allows for wider variety of shades by taking the angles from which the light approaches the object as well as the bounce of light of an object behind the initial object into account as the Figure 2.9(b)below shows[37].



(A) Screen Space Directional Occlusion Results. Source: [37]



(B) Yellow surface light bounce to the floor. Source: [37]

FIGURE 2.9: Screen Space Directional Occlusion different rendered results

2.4.4 Comparison

In this part we present a comparison between the different ambient occlusion computation methods that has been introduced in the above sections see Table 2.1.

	Methods	Advantages	Disadvantages
Object Based	- Ambient Occlusion Fields	- Efficient for shadow computation	- Rigid mesh are the only shadowed
	- Fast Precomputed AO for proximity Shadows	- On the fly implementation - Fast computation execution - Occluder complexity independency	- Memory overhead
Point Based	- Dynamic AO and Indirect Lighting	- Fast for Ambient Occlusion computation	- Polygonal data treatment
	- Hardware Accelerated AO computation	- Easy to implement - No preprocessing	- High light adaptation for models
Screen Space Based	- Screen Space Ambient Occlusion	- Zero loads time - No preprocessing - No cpu usage	- High frequency Noise
	- Screen Space Directional Occlusion	- One bounce of indirect lighiting is included	- Bad classification - Only visible shippers can contribute to indirect illumination

TABLE 2.1: Ambient Occlusion different methods comparison

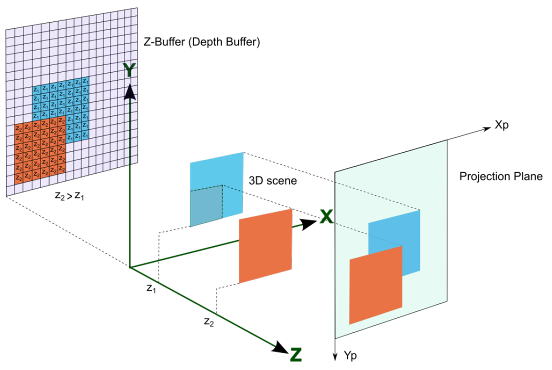
2.5 Ambient Occlusion in Screen Space

Each of the previous mentioned techniques in Section 2.4 has a different understanding of the Ambient Occlusion factor calculation, we have chosen the screen space ambient occlusion where we go through its foundation and its different techniques that use the same pillar which is the depth buffer to compete for a better realism enhancement.

2.5.1 Definition

Screen Space Ambient Occlusion is a technique that depends on the depth buffer that is defined as a grayscale texture like in Figure 2.10(a) as a visual geometry location approximation as Figure 2.10(b) shows which used to define the occluded areas that has a low light intensity contribution like corners, crack, or even human body wrinkles which uses alongside with the depth buffer a per-vertex normal and position buffers where all these three combined together to achieve an approximated ambient occlusion and more realistic lighting enhancement that can be generated on the fly with zero load time where the ambient occlusion factor can be approximated by sampling the surroundings of each depth value that is stored as pixel and can be retrieved from the depth buffer considering the surroundings samples as a finite quantity where they are generated using the powerful monte carlo integration techniques[28].

(A) Scene geometry approximation using the depth buffer. The grayscale values defines geomtery location where black values for closer geometry and white values for far geometry.



(B) Depth buffer illustration Source: [38]

FIGURE 2.10: Depth buffer visualization and its geometric definition

2.5.2 Screen Space Ambient Occlusion Techniques

Screen Space Ambient Occlusion variate through a lot of techniques in this Section 2.5.2 we come across a various techniques of the Ambient Occlusion Approximation using the screen space as calculation field.

2.5.2.1 Crytek Ambient Occlusion

Crytek Occlusion is considered as the first Screen Space based Ambient Occlusion approximation technique that has been introduced by Crytek in 2007[36] as a scale down version of the ambient occlusion in *Crysis* as we have seen in the previous Figure 2.8 which has been implemented in Crytek game engine that is the *Cryengine 2* where as mentioned in Section 2.5.1 that it uses the depth buffer as depth sampling inside a sphere around the occluded point which is presented by a pixel where we calculate the ambient occlusion factor at by a simple comparison between the original point depth value S_z and the depth of a sample that is projected into the scene S_d like Figure 2.11 illustrates after randomly choosing the kernel sample resulting the ambient factor approximation which gives much more realism into the scene.

Where the original base of all the SSAO techniques is giving by the following formula:

$$AO \approx \frac{1}{N} \sum_{N}^{n=1} V'(S_n)$$

With:
N: as the sample kernel which is the number of used samples.
S_n: is a sample point that has random position around the center of the sphere p where we want to calculate the SSAO at.
V': is the same ambient occlusion visibility function but with a simplified calculation that is done between pixel depth rather than geometry like in AO, which is given as follow:

$$V'(s) = \begin{cases} 1, & S_d > S_z \\ 0, & \text{Otherwise} \end{cases}$$

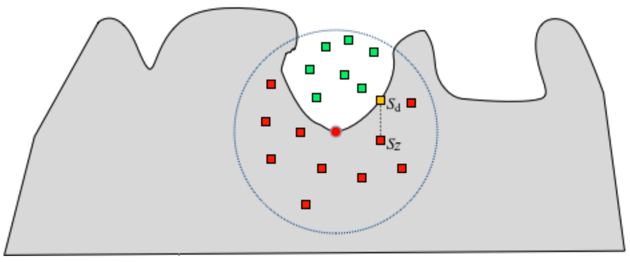
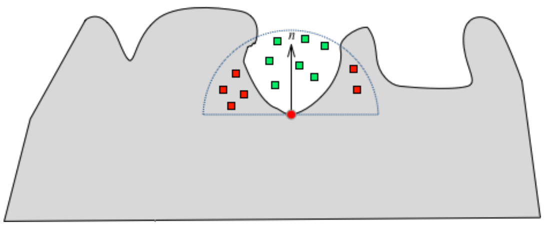


FIGURE 2.11: Ambient Occlusion factor approximation using sphere sampling in Crytek technique, with the squares representing pixels as the green defined the visible samples and red defines the occluded ones

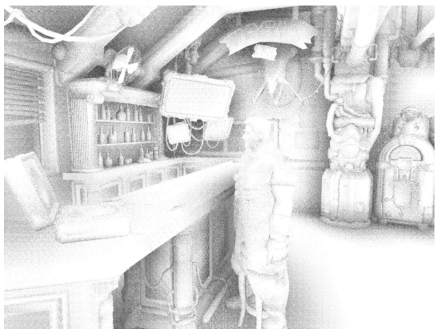
2.5.2.2 StarCraft 2 Ambient Occlusion

Regardless the great results of the CrytekAO it is quite obvious that it will approximate the ambient in surfaces where there is no occluded areas at all like flat surfaces which represent a wall for examples because of the sphere sampling method which cause inside geometry to contribute in the AO factor approximation resulting a darker color in each of these flat surfaces.

StarCraft2 has been introduced in 2008[39] in *BLIZZARD entertainement* game development company which defines the sample kernel as sampling in a normal-oriented hemisphere that is illustrated in Figure 2.12(a) where samples considered to be alongside the sampled point rather then behind it inside the geometry resulting better AO approximation as we can see in Figure 2.12(b) flat surfaces are white rather then darker like in the CrytekAO used method.



(A) StarCraftII samples distribution illustration which is considered as a distribution with a normal-oriented hemisphere.

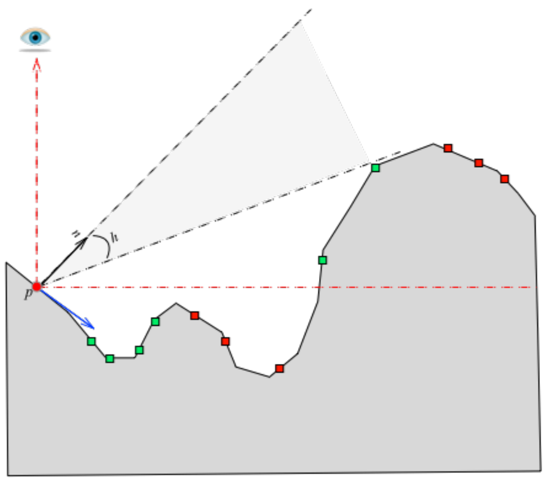


(B) SSAO calculation using random sampling. Source: [39]

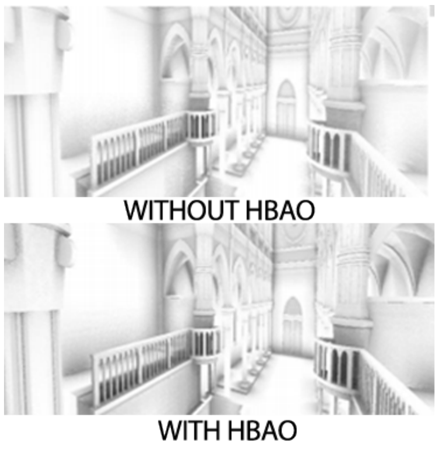
FIGURE 2.12: StarCraftII rendered results using the normal-oriented hemisphere

2.5.2.3 Horizon Based Ambient Occlusion

Horizon Based Ambient Occlusion or HBAO is a technique that has been introduced in 2008[40], that uses the depth buffer in order to ray marching which is obviously in screen space to find a visible horizon angle h[40] as Figure 2.13(a) illustrate by setting a perpendicular ray to the observer view direction which makes sampling achieved along side this perpendicular ray where in each step the depth value is obtained and compared to the previous depth value to check if it is inferior which leads to update the horizon angle, several rays are cast and the ambient occlusion factor is then approximated using the average free horizon angle which makes scene rendering looks more realistic than before as Figure 2.13(b) shows.



(A) Horizon Based samples distribution using Ray Marching and technique illustration with the average free horizon angle as the last visual contribution.

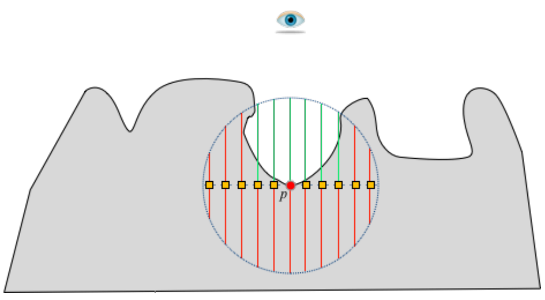


(B) Ambient Occlusion without and with shadows respectively. Source: [40]

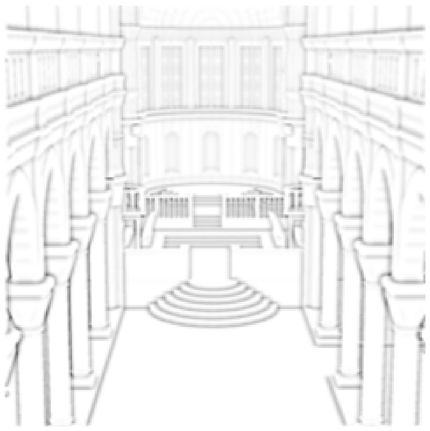
FIGURE 2.13: Horizon Based Ambient Occlusion technique result and illustration

2.5.2.4 Volumetric Obscurance

One of the screen space methods are the volumetric obscurance that depends on the 3D neighborhood volume around p[41]. Where the ambient occlusion factor is approximated using the unoccluded and occluded 3D volume with the sphere sampling method taking as a center the point p where we want to approximate the ao factor and a random generated samples which are perpendicular to the observer view direction with each sample represent small volumetric piece that is defined as line segment as Figure 2.14(a) illustrate which can be occluded or unoccluded where then the depth buffer is used to project each sample in other words the line segment with applying the simple trigonometry to calculate the visible samples in order to determine the AO factor as Figure 2.14(b) demonstrate.



(A) VO used technique for sampling in a sphere with the green and red line segments are the occluded and unoccluded parts of each sample



(B) Ambient Occlusion approximation using the Volumetric Obscurance technique rendered results. Source: [41]

FIGURE 2.14: Volumetric Obscurance technique rendered results and technique illustration

2.5.2.5 The Alchemy Ambient Obscurance

The Alchemy Ambient Obscurance is a technique that uses the falloff function rather than the occlusion function it was introduced in 2011[42] by *NVIDIA* and *Vicarious Visions Studio* where it has been implemented in their Alchemy game engine.

We mentioned in Section 2.3 that ambient obscurance is used to approximate the ambient occlusion that is used rather then the ambient obscurance name for its familiarization and popularity in using screen space to approximate the indirect lighting.

The falloff function ρ that represent the Ambient Obscurance is given by the following formula:

$$\rho(d) = \frac{u \cdot d}{max(u,d)^2}$$

Where:
d: is the sample distance.
u: is a user-specified parameter to choose the exact shape (intensity scale).

The AlchemyAO uses AO inversion rather than direct computation which means with higher d_{max} the attenuation function ρ will converge toward 0 and not 1 that we mentioned in Section 2.3.

We have from inversion the AO function:

$$AO = 1 - \frac{1}{\pi}\int_{\Omega} \rho(d)(n \cdot \omega)d\omega$$

Which leads to:

$$AO = 1 - \frac{u}{\pi}\int_{\Omega}\frac{d\cdot(n\cdot\omega)}{max(u,d)^2}d\omega$$

With the user-specified parameter u outside the integral then a v vector which equals to $v = \omega \cdot d$ is defined leading the formula to be simplified as follow:

$$AO = 1 - \frac{u}{\pi}\int_{\Omega}\frac{\vec{v}\cdot\hat{n}}{max(u,d)^2}d\omega$$

While each of u and d which are the user specified parameter and the distance from point to sample respectively are assumed positive it is quire assertive that $max(u,d)^2 = max(u^2,d^2)$ and during the fact that $v\cdot v = |v|^2$ is the same as d^2 which is caused v being defined as the vector between each ray direction ω and point p like it is illustrated in Figure 2.15(a) simplifying the formula to:

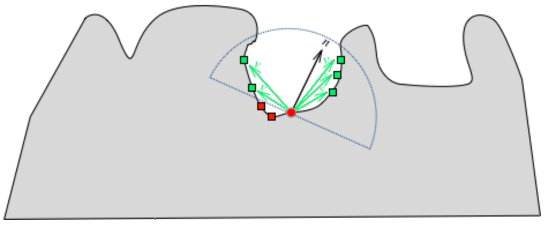
$$AO = 1 - \frac{u}{\pi}\int_{\Omega}\frac{\vec{v}\cdot\hat{n}}{max(u^2,\vec{v}\cdot\vec{v})}d\omega$$

Considering the integral as a continuous space which can't be calculated using a machine during that an infinity of ray direction can contribute the computation. A finite quantity of samples is used alongside with the powerful monte carlo integration in order to get the AO factor ready to approximated with the following formula:

$$AO(p) = max\left(0, 1 - \frac{2u}{N}\sum_{m=1}^{N}\frac{max(0,\vec{v_m}\cdot\hat{n}+Z_c\beta)}{\vec{v_m}\cdot\vec{v_m}+\varepsilon}\right)^k$$

Where:
N: is the number of kernel samples.
n: is the surface normal.
v_m: is the sampler vector which is calculated using (S_m - p).
β: is the depth bias.
Z_c: is the occludee's depth value.
u: is the user-specified parameter which is also known as the intensity scale.
k: is the contrast controller.
ε: is a small value to prevent 0 division.

The AlchemyAO leaves the Ambient Occlusion factor realism enhancing tuned using the four previous mentioned parameters which are the radius, bias, intensity scale and the contrast value which result a good ambient occlusion approximation as seen in Figure 2.15(b).



(A) Alchemy Ambient Occlusion Technique illustration where the sample are in a normal-oriented hemisphere with v a vector between the sample point and the potential occluded point.

(B) Alchemy Ambient Occlusion in rendered scene. Source: [42]

FIGURE 2.15: AlchemyAO Alchemy engine results and technique illustration

2.5.2.6 Comparison

In the Table 2.2 a comparison between the different screen-space techniques in approximating the ambient occlusion.

Technique	Avantages	Disadvantages
CrytekAO	- Fast results. - Simple to implement.	- Spherical Sampling problem with flat surfaces.
StarCraftIIAO	-Hemisphere sampling (Improved sampling).	- Self-occlusion. - Edge case.
VolumetricAO	- Better volume approximation (due the line samples). - No blur required with 32 samples usages. - Suited for dynamic scenes.	- Low performance when it comes to area samples. - Absence of thin objects influence.
HBAO	- Better resulted shading (due the heightfield assumption).	- Randomized ray directions (due the usage of random vectors.
AlchemySSAO	- Intuitive - Efficient - Simple to implement - Artistic control - Robust	- Bilateral filtering causes low performance. - Sample variance. - Under-occlussion.

TABLE 2.2: Comparison between the different SSAO techniques.

2.6 Filtering Techniques

Approximating the Ambient Occlusion in the screen space using random depth samples produces the banding effect due the textures usage which is very visual and noticed with the real human eye which can be fully cleared using a rotation technique around the z-axis producing as a result a less noticeable effect known as *Noise* which can be reduced using filtering techniques which in most cases uses interpolation in order to enhance the Ambient Occlusion approximation screen space techniques that we go through in this part.

2.6.1 Anisotropic Filter

Giving a rendered scene a realistic aspect inherit from the real world is quite computational without using textures which make the scene looks more detailed by using several different textures like colorization texture, transparency, reflectively and bumps that are mapped to an object and processed by the GPU to achieve the realistic appearance, where the distance between the camera and the object texel which is a pixel of a texture in other words affects the observable level of details that consist the usage of mipmap which a is duplication of the master texture that is used often in a repeated pattern like brick walls or floors where the repeated process gives a blurring effect or visual artifacts as the mipmap are far and has an angle view with the camera multiple mipmaps are sampled for a single texel.

Anisotropic filtering is one of the techniques that is used avoid the blurry or artifacts effects by providing a superior quality in scene rendering at a low cost of performance where it has a high frequency GPU usage due the mipmap height and width scaling with a power of two level from 2x to 16x providing a high quality of textures and clarity as the level goes up as Figure 2.16 shows[43].

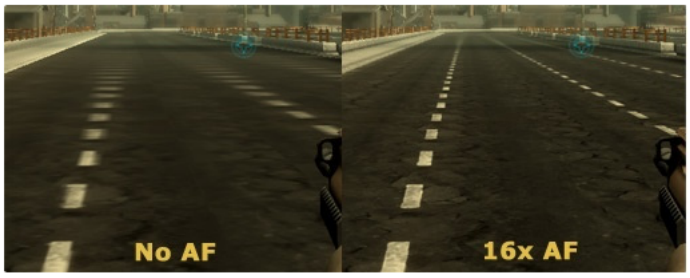


FIGURE 2.16: The huge difference between with and without Anisotropic filter as the distance goes higher. Source: [44]

2.6.2 Anti-Aliasing

Anti-Aliasing is defined as rendering technique that is used in order to minimize the spread of aliasing which is a visual artifact that looks like the steps on a stair as Figure 2.17 illustrate, on any smooth or non-perpendicular surface of 3D object which is caused by a resterization stage of the graphic pipeline or from transforming a vector image like Encapsulated PostScript (EPS) for example to a raster image like Bitmap (BMP), that leads to create visible inconsistencies in edge continuity due to the GPU only coloring a pixel if the line passing through it occupies more than half of the space, resulting jagged edge where we would normally expect smooth contiguous line[43].

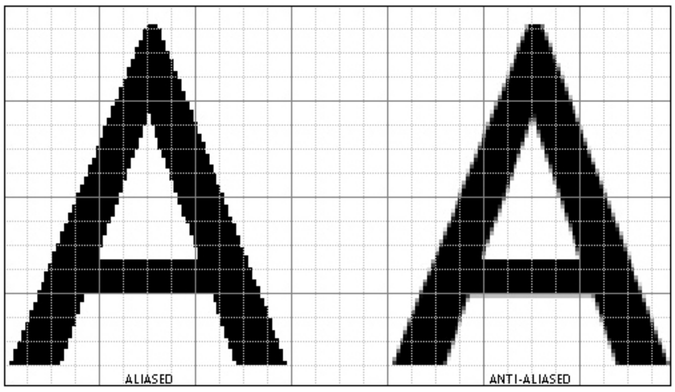


FIGURE 2.17: The effect resulted by the aliasing and its enhancement using the anti-aliasing technique. Source: [43]

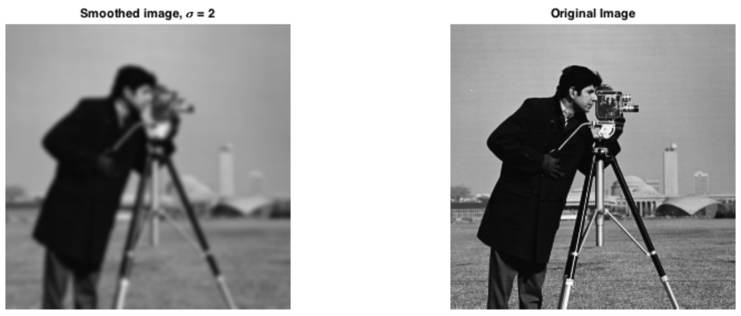
2.6.3 Gaussian Filter

Image processing has always included filtering techniques which provide a much more images enhancement due the fact that a passage from continuity interval to discrete interval or randomness usage must be done in computer graphics which leaves us with a quite amount of noise that can be reduced by smoothing it which is a known effect nowadays under the name *blur* as Figure 2.18 illustrate using the Gaussian filtering that is defined as a weighted average of the adjacent positions intensity [45].

Where the Gaussian filter is giving by the following formula:

$$g(x,y) = \frac{1}{2\pi\sigma^2} \exp - \frac{(x^2+y^2)}{2\sigma^2}$$

With:
(x,y): is the image coordinates.
σ^2: is the variance of the Gaussian filter respectively σ as the standard deviation.



(A) result of camera man with Gaussian filter (B) Without Gaussian filtering Original Image

FIGURE 2.18: Gaussian Filtering illustration using Matlab platform

2.6.4 Bilateral Filter

Like the Gaussian filter the Bilateral filter technique that is used to smooth images as Figure 2.19(a) shows results of applying the bilateral filter but without loosing edges as it is illustrated in Figure 2.19(b) and considered to be a non-linear technique where it has been introduced in 2009[46] it is actually based on the Gaussian filter technique when it comes to the weighted average on intensity values from nearby pixels for pixel intensity replacement.

Where the Bilateral filter is defined using the following formula:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(||p-q||) G_{\sigma_r}(|I_p - I_q|) I_q$$

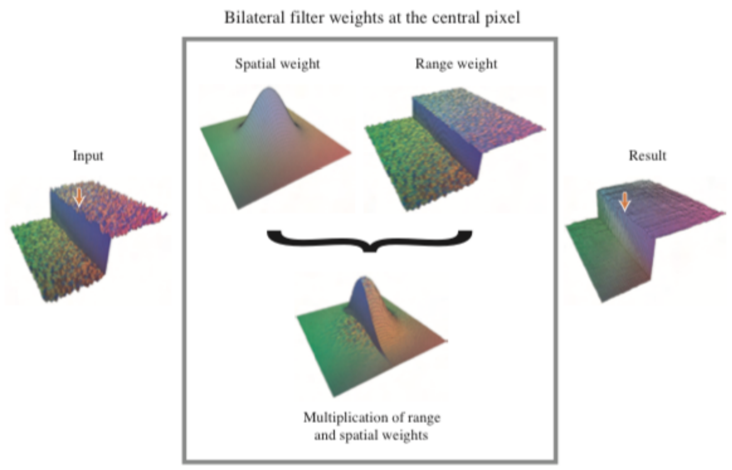
With:

$\frac{1}{W_p}$: as a normalization factor which is new comparing to the Gaussian filter.

$G_{\sigma_s}(||p-q||)$: as a space weight which used for the weighted average like the Gaussian filter.

$G_{\sigma_r}(|I_p - I_q|)$: as a range weight which makes the weighted average depends on the image content.

(A) Blurring effect results using the bilateral filter by taking an image as an input and convert it into blurred output smoothed in other word. Source: [46]


Bilateral filter weights at the central pixel
Spatial weight
Range weight
Input
Result
Multiplication of range and spatial weights

(B) As we can see the bilateral filtering technique combine between spatial weight and range weight in order to avoid loosing edges and gives great results when it comes to blurring effect. Source: [46]

FIGURE 2.19: Bilateral Filtering results and technique illustration

2.6.5 Comparison

After we came across the different filtering techniques a simple comparison between them are given in the Table 2.3 below.

Filtring Method	Advantages	Disadvantages
Anisotropic Filtering	- Optimal Visual Quality. - Eliminate all kinds of Aliasing Effects.	- Memory overhead.
Anti-Aliasing	- Has a maximum details saving. - Efficient Implementation using Modern Material.	- Some Artifact can't be treated.
Gaussian Filter	- Noise Reduction. - Separable Filtre. - Symmetrical in rotation for large filters.	- Details disappearing caused by high Variance value. - Takes a quite amount of time.
Bilateral Filter	- Noise suppression. - Edges preserving. - Resulting smoothed images. - Easy to understand (Weighted mean of nearby pixels). - Easy to adapt (Distance between pixel values). - Easy to set up (Non-iterative).	- Absence of impulsive Noise elimination. - Filtering replacement in noisy or non-noisy cases. - It softens the image alone, does not sharpen it.

TABLE 2.3: Comparison between the different filtering techniques used in the computer graphics

2.7 Conclusion

During this chapter we uncovered the power of different creative methods that can approximate the ambient occlusion which uses different aspects and knowledge that compete in order to achieve a more realistic global illumination approximation alongside with their most known advantages and disadvantages that could be enhanced. Also we discussed noise and aliasing filtering technique which can result a quite good smoothness that is known as blur to reduce the amount or even eliminate the visual artifacts which the human being eye can distinguish along with a summarized comparison that recaps their differences when it comes to time, memory and result efficiency.

Interactive Ambient Occlusion achievement leads to quality reduction which is obviously caused by the approximation that is an important obligation in order to have an acceptable results when it comes to Real-Time obtaining which is the essential requirements for interactive applications like videogames.

The greatness of screen space allows us to approximate the ambient occlusion in real-time with the screen space ambient occlusion method that is a less consuming method when it comes to time or performance and it can be acclimated easily in computer graphics applications. Though its realistic results in visualization enhancement it can produce a lot of noise due to randomness usage which can be resolved using filtering techniques.

Chapter 3

GPUs and OpenGL Evolution

3.1 Introduction

Ambient Occlusion either calculation or approximation that has been discussed in second chapter has no meaning if it is not rendered in real-time which is requires for interactive applications most often in videogames. the CPU by itself cant handle all the geometry that can be defined for a videogame because of all the mathematical formulas that must be solved from the simple normalizing formula to the complicated illumination models, where the GPU here comes in handy using its special gift that is the parallelism through its multiple threads a lot of calculation can be done in a fraction of a seconds. In this chapter we came across the different between the CPU and the GPU along with its parallelism when it comes to acceleration with the architecture behind the GPUs and the way that are programmed with in order to create realistic rendered images in real-time, after that we have been introduced the OpenGL and its evolution from the old to the new one along with the used structures for shader programming.

3.2 GPUs versus CPUs

In a simple system or machine for other word, there may be only one processor which is the central processing unit (CPU), which it handles each of the normal and graphical processing. The main graphical function of the processor is to take two things into account the first one is to take the shape of the graphical primitive as a specification like lines, circles, polygons and the second thing is to assign values to the pixels in the framebuffer that is directly addressed by the CPU in the old days graphics systems which is in other word the displaying window where the graphical function must figure out how to transform geometric shapes into colored and located pixels[4].

The GPU which is an abbreviation of *graphics processing units* has came to take responsibility of the graphical functions which is can be either embedded directly with the mother board or on a graphics card which is usually used with desktop where framebuffer became accessed through the GPU and each of them are located on the same circuit board for faster assigning[4].

GPUs has evolved to the point where they are as complex or even more complex then CPUs and become so powerful to even be used as mini supercomputers.

3.3 Acceleration Using GPUs

As the GPU had more threads than the CPU it became parallelism supported which is extremely beneficial with independent graphical operations causing a huge acceleration into rendering three dimensional shapes comparing to the cpu that can only preform sequential operations.

A Fixed Function pipeline was the first rendering procedure that has been out and defined as the rendering stage not being controllable which means that the rendering functions and geometry manipulation was embedded (built-in) directly inside the GPU[47].

3.4 GPUs Evolution

Graphics on desktop computers were handled by *Video graphics Array* (VGA) where it is defined as a memory controller attached to the DRAM and display generator with a specified main function that is receiving image data, arrange it properly, and send it to a video device (monitor)[48].

Different graphics acceleration components are added to the VGA controller by the 1990s for triangles rasterization, texture mapping and simple shading leading NVIDIA to release the "GeForce 256" that is showed in Figure 3.1 and marketed as the world's first GPU[49].

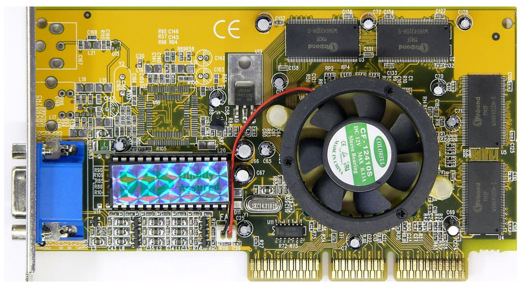


FIGURE 3.1: NVIDIA Geforce 256 first GPU. Source: [48]

The original GPUs were modeled from the idea of a graphics pipeline which combine between GPU(GPU cores) and CPU(Opengl, Direct and with a main task which is transforming a 3D coordinates into 2D screen pixel coordinates[48].

GPU came into existance for graphical purpose, it has now evolved into computing, accuracy and performance. The fast computation of the GPU over the last years has open a new world of possibilities for high-speed computation where graphics cards are widely used for accelerated rendering of three dimensional scenes and in the field of image processing[49].

GPUs can be set up in a wide range of devices from desktops and laptops to mobile handsets nad super computers. Thanks to their parallel structure, GPUs implement a variety of 2D and 3D graphics primitives processing in hardware[48].

3.5 GPU Architecture

Due to the phenomenal tasks that can a GPU do an architecture for rendering achievement are required which are presented in the next sections.

3.5.1 Material Components

Achieving the acceleration aspect using the GPU needs a lot of memory and processors with adding chipsets and registers to ensure communication between them in order to the GPU to stay synchronized and give great results when it comes to parallelism.

As the GPU being a processor unit means that it is like the Central Processing Unit (CPU) a single-chip processor with a much bigger difference than the CPU the GPU may have hundreds of threads (Cores) while the number is limited in CPU that is capable to be executed at the same time to do independent computations with a VRAM which is the *video ram* that can reaches 4Go or Higher nowadays [48].

3.5.2 Shaders

Manipulating geometry using the CPU which means just OpenGL can't be freely as using GPU due to the complex geometry and phenomenal rendering techniques that can be achieved with the GPU particularly using the *Shaders* which are defined as a small programs executed within the GPU to render shapes into the screen passing by several steps:

⇒ Scene objects Modeling.

⇒ Objects vertices input for Vertex Shader.

⇒ Transform Space Coordinates to Screen Coordinates

⇒ Rasterizing the 3D geometry into a pixel output for the Fragment Shader.

⇒ Colorization of pixels based on the giving color or texel value.

⇒ Projected scene into the framebuffer for visualization.

With the following Shader types list:

⇒ **Vertex Shader**: Which is responsible for per-vertex operations that takes geometry vertices as an input.

⇒ **Tessellation Shader**: that is responsible for surfaces subdivision into smaller primitives for smoother meshes.

⇒ **Geometry Shader**: that is specialized in suppression vertices or primitives which is known as geometry operations.

⇒ **Fragment Shader**: Pixel Colorization requires the fragment shader to determine the final color of pixels.

While the Shader being small programs executed within the GPU as mentioned above it requires a communication variables between the host part which is the CPU and GPU and between the shaders themselves some variables are used:

⇒ **Attribute**: which is per-vertex informations which is the same as **In** in modern OpenGL.

⇒ **Uniform**: a Global variable passed from the client side to the GPU and shared between the shader programs.

⇒ **In**: for input that defines a shader input that comes from a previous stage of rendering.

⇒ **Out**: for Output which is used to pass a computed result to the next stage of rendering.

⇒ **Const**: Local constant for shaders which has a fast accessing due to the constant memory space reservation.

Each of **In** and **Out** are used in modern OpenGL they are the replacement of the keyword *Varying* in older OpenGL version.

3.5.3 Graphic Pipeline

In order to render 3D scene into a virtual sized screen several steps are demanded which make three dimensional (3D) coordinates conversion to two dimensional (2D) coordinates where the conversion is meaning rasterizing primitives shapes like polygons from continuous space which is world space into a discrete space that is the screen coordinates that are known under the keyword *pixels*, where in the first years of computer graphics a Fixed function pipeline was used in order to display rasterized results into the frambuffer which means that the developer has no possibility in modifying the pipeline that lacks to flexibility and understanding the way that graphic cards works as the Figure 3.2 illustrate the fixed function pipeline.

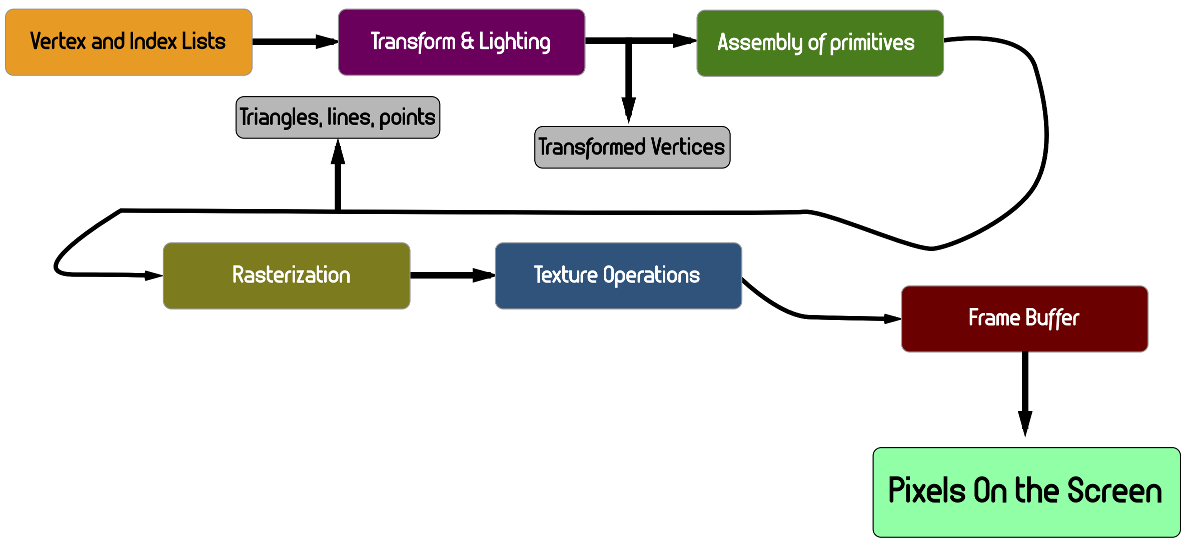


FIGURE 3.2: First Graphic Pipeline that is known as the Fixed function pipeline.

Fixed function pipeline contains only fixed sequence of processing stages which means that are unmodified from users and can only be configurable which makes creativity and flexibility absence when it comes to reality simulation or even imagination creation, where the graphics card nowadays are more flexible than the old versions by providing a large programmable parts of the graphic pipeline than before.

3.5.4 Graphic Pipeline Evolution

The graphic pipeline has been evolved considerably since its introduction that was discussed in the section above 3.5.3 by providing more flexibility and better understanding to the way rendering is working which is preformed by a rasterization process and that is done by adding four programmable stages that are the Vertex Shader, Fragment Shader which are the shaders that must be included additionally Geometry and Tessellation Shaders as optional choice, where the Figure 3.3 below illustrate the new graphic pipeline.

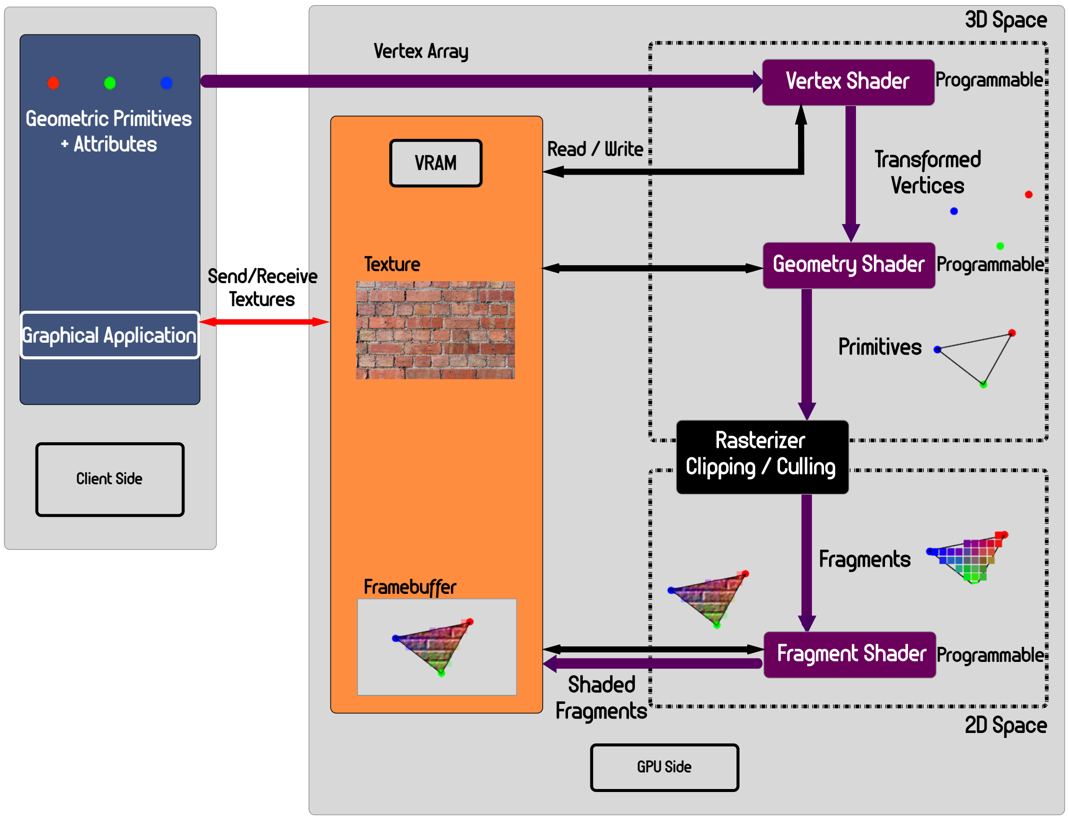


FIGURE 3.3: Modern Graphic pipeline illustration.

The graphic pipeline begins with the geometric data that is provided (vertices and geometric primitives) and first processes it through a sequence of shader stages: vertex shading, tessellation shading (which itself uses two shaders), and finally geometry shading, before it's passed to the rasterizer. After that the rasterizer will generate fragments for any primitive that's inside of the *clipping region*, and execute a fragment shader for each of the generated fragments[3].

Shaders play an essential role in creating graphical applications specially when it comes to creativity and flexibility and a better understanding on how virtual scenes are really achieved.

3.6 GPUs Programming Languages

graphic pipeline programmable stages requires to be written by their user in order to do a task that is provoked through a draw call, using a high-level C-like language nowadays we can write any programmable stage in order to be executed where the written shaders is attached to a program that is called any time we want to use a specific shaders. In the next sections some of the popular gpus graphical programming languages are discussed.

3.6.1 OpenGL Shading Language (GLSL)

GLSL which refers to Open**GL** **S**hading **L**anguage is in the class of languages that can be considered "C-like" as mentioned in the section above 3.6 which means that each of GLSL and C are almost the same when it comes to GLSL syntax and model with some differences that make it more suitable for graphics and parallel execution like built-in mathematical types like vector or matrices that was inhirit from the *RenderMan* Shading language for the deferred time rendering while being the GLSL specifically designed to *OpenGL* standard API and to be run on massively parallel implementations[47].

3.6.2 High Level Shading Language (HLSL)

HLSL for **H**igh **L**evel **S**hading **L**anguage that is the same as GLSL but designed for Microsoft *DiretX* rather than OpenGL which was developed in collaboration with NVIDIA that was also influenced by the syntax and philosophy of the C programming language and also included elements from the Renderman shading language[49].

3.6.3 C for Graphics (CG)

C for **G**raphics was introduced by a collaboration between Microsoft and NVIDIA while working on HLSL which is a programming language for the *DirectX* where **CG** is considered to be a cross-platform programming language which means that it can be used with the two fancy graphics APIs: OpenGL and DirectX[49].

3.7 OpenGL Evolution

Real-time rendering has became possible thanks to OpenGL which is defined as an *Application **P**rogramming **I**nterface* which is just a software library for accessing graphics hardware feautures that has been introduced at *Silicon Graphics Computers Systems* and their IRIS GL where **GL** stood for (and still stands for) "Graphics Library" with Version 1.0 released in June of 1992[47], it is designed as a streamlined, hardware-independent interface that can be implemented either on different graphics hardware systems, or entirely in software due to the absence of graphics hardware with an operating system windowing system usage due the fact that OpenGL doesn't include functions for performing windowing tasks or processing user input[3].

OpenGL is implemented as a client-server system, with application side as a client and the OpenGL implementation provided by the manufacture of the graphics hardware being the server where it doesn't provide three dimensional models loading functionalities as long as the absence of image files reading which leads to the need of *Libraries* usage in order to do such things[3]. The various versions of OpenGL are presented in the Table 3.1 below.

Version	Publication Date
OpenGL 1.0	January 1992
OpenGL 1.1	January 1997
OpenGL 1.2	March 1998
OpenGL 1.2.1	October 1998
OpenGL 1.3	August 2001
OpenGL 1.4	July 2002
OpenGL 1.5	July 2003
OpenGL 2.0	September 2004
OpenGL 2.1	July 2006
OpenGL 3.0	August 2008
OpenGL 3.1	March 2009
OpenGL 3.2	August 2009
OpenGL 3.3	March 2010
OpenGL 4.0	March 2010
OpenGL 4.1	July 2010
OpenGL 4.2	July 2011
OpenGL 4.3	August 2012
OpenGL 4.4	2013
OpenGL 4.5	2014
OpenGL 4.6	2017

TABLE 3.1: OpenGL Versions and Publication Dates.[47]

The list below briefly describes the major operations that an OpenGL application would perform to render an image:

⇒ Data Specifying for shapes constructing using OpenGL's geometric primitives

⇒ Shaders usages to perform computations on input primitives for position, color and other attributes determination.

⇒ Fragment Shader execution for each fragment generated by rasterization leading to determine fragment's final color and position.

⇒ Visibility and blending per-fragment tests can be additionally performed.

In 2008 the ARB which stands for **A**rchitecture **R**eview **B**oard decided to sperate the OpenGL specification into two profile *Compatibility* and *Core* profiles along side the release of the OpenGL 3.2 due to drivers bugs caused by the difficulties of specifying the interaction between older legacy features in a graphics card and the new recent features[47] after the 3.1 version has removed all the functions that was deprectaded in 3.0 version like: (Fixed function pipeline, glBegin/glEnd, etc) that are only configurable and doesn't provide an understandng on how OpenGL itself works.

3.7.1 Compatibility profile

The *Compatibility profile* defines keeping all old legacy features which are known as the "deprecated functions" of OpenGL in other words it maintains the backwards compatibility with all revisions of OpenGL back to version 1.0. which means that an old written software in 1992 should compile and run on a modern graphics card with higher performance than the first time the software has been produced. Although it has been really exist to allow software developers to maintain legacy applications and to add features to them without having to tear out years of work in order to shift to a new API[47].

3.7.2 Core profile

Core profile from the other hand removes a number of legacy features that has been point to them as the "deprecation functions" leaving those that are truly accelerated by current graphics hardware. This specification is more shorter that the compatibility version where it is strongly recommended for better OpenGL experience and understanding and for those who demand more freedom and flexibility[47].

3.7.3 Structures for Shader Programming

Using Modern OpenGL requires using modern methods in order to pass object data as attributes like: (positions, colors, texels coordinates) and so on anything that is related to the object data passed to the shaders from the graphical application in order to be processed and rasterized after than displayed in windowed screen called the framebuffer using the operating system windowing system, where a set of a bindable buffers are used in order to accomplish such a task conversely the old OpenGL (compatibility profile) where the geometry shapes which are the 3D models that are used in a 3D scene are loaded into the scene itself using the deprecated functions like glBegin,glEnd by drawing a set of triangles attached with each other presenting the geometry shape that is known as the three dimensional "*Model*".

3.7.3.1 Vertex Buffer Object (VBO)

The **V**ertex **B**uffer **O**bject is defined as a memory portion on the GPU that contains all the vertices data from position to texture coordinates or other attributes with telling OpenGL how to interpret the memory and specifying the data send to the GPU where the vertex shader deal with each vertex providing a transformed vertices to the geometry shader if it is implemented or directly as primitives to the rasterizer. sending data using VBOs rather than CPU makes data accessing faster due the fact that we can send a large portion of vertices data to the GPU rather than sending data per-vertex at a time [50].

3.7.3.2 Element Buffer Object (EBO)

An **E**lement **B**uffer **O**bject is the same as the Vertex Buffer Object but rather than using the vertices themselves to draw call a list of indices are used instead which has the vertices arrangement for drawing calls where the benefit of the EBO is that it removes duplicated vertices for each face by providing a unique definition of the vertex data and an index list to tell the vertex shader how it should interepret the vertex input[50].

3.7.3.3 Vertex Array Object (VAO)

A **V**ertex **A**rray **O**bject from the other hand holds all calls that is related to a vertex attribute that is stored inside a VAO which makes the advantage of configuring the VBO or EBO only once rather than re-configuring it each time we need to use a specific vertex attribute leading to make the process of vertex data and attribute configurations which are implemented with the VBO or EBO changing easy and on the fly only by binding different VAO[50].

3.7.3.4 Frame Buffer Object (FBO)

The **F**rame **B**uffer **O**bject is as all the buffers has a memory portion which holds a color buffer for color values and a depth buffer for depth informations, a stencil buffer that allows us to discard some fragments based on particular condition as in Figure 3.4 the combination of the framebuffer are illustrated where a FBO is defined either by a default framebuffer which is created with the windowing process or a personal framebuffer that is created by users and offered by the provided flexibility of the modern OpenGL[50].

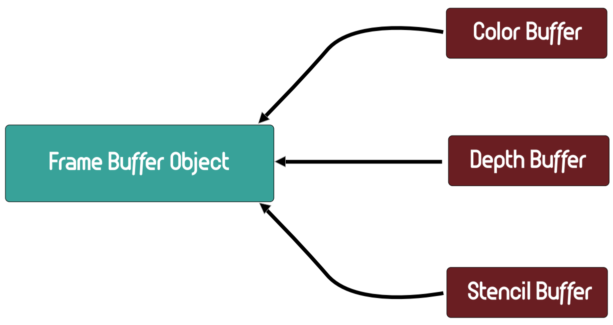


FIGURE 3.4: FrameBuffer combination of the color,depth,stencil buffers.

3.7.3.5 G-Buffer

The G-buffer with **G** standing for Geometry is defined as storing geometry informations into textures for later use which are commonly used in deferred shading for performance optimization and more faster results than the forward shading, where the G-buffer can contains:

⇒ Vertices Positions.

⇒ Surfaces Normals.

⇒ Diffuse Intensity.

⇒ Texture coordinates.

⇒ Specular Intensity.

The following Figure 3.5 illustrate some of the gBuffer rendered textures.

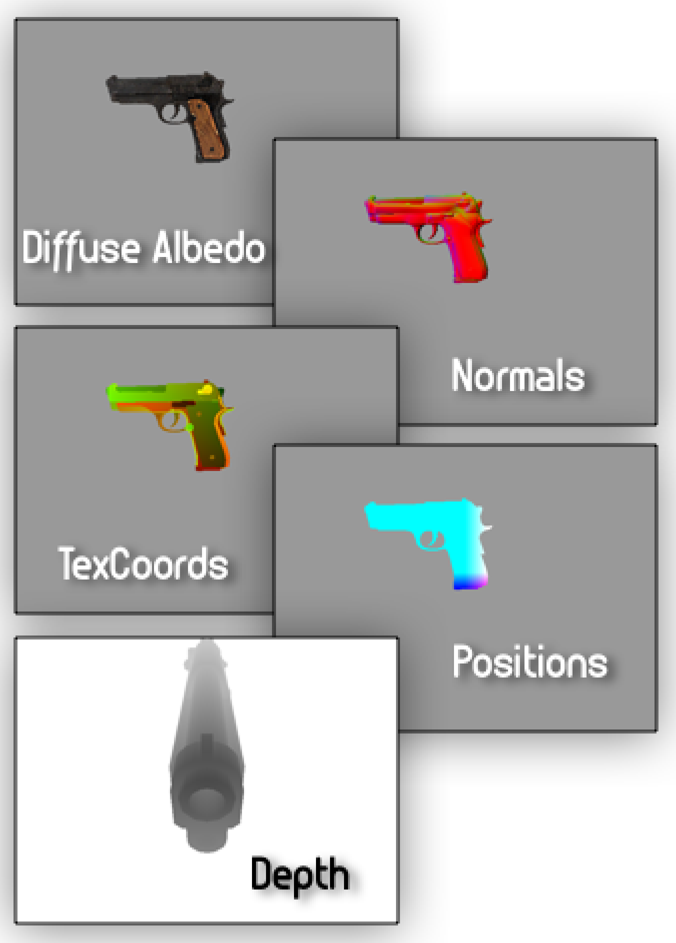


FIGURE 3.5: Geometry informations Rendered for G-Buffer Textures for later use.

3.8 Conclusion

Through this chapter we have introduced the essence of computer graphics in real-time which is based on the GPU and shaders programming that let users to be more creative than before as long as the graphic pipeline evolution from its beginning to the used models nowadays where these shaders needs a high-level languages when it comes to coding like the **GLSL** that is designed specifically to OpenGL and **HLSL** for the Microsoft DirectX after that we went deeper into the details with the distinguish between The *Compatibility* and *Core* OpenGL profiles and the useful data structures which are obligated in the *Core profile* type of OpenGL that are encapsulated inside a *VAO* which is a data structure by itself for re-using and easiness and lastly we have defined the powerful rendering tool which is the *G-Buffer* that is dedicated for deferred shading and performance optimization thanks to the OpenGL flexibility in new versions.

Chapter 4

Results and Implementations

4.1 Introduction

Through this chapter we have begin by the description and the objectives of the project alongside with the general design of the application continuing after that with its realization where we start by defining the used materials and softwares that has an influence on the project achievement along with the application structure, all this part part combined allowed us to achieve the objectives of the project where we present its results in the last part of this chapter.

4.2 Project Description and Objectives

One of the main objectives of this project is to implement the Alchemy Screen Space Ambient Obscurance technique which has robust result in approximating the Ambient Obscurance in real-time that add a quite amount of realism into the scene where sampling and filtering are the fundamentals of this achievement, but due the fact that it uses randomness in samples providing and bilateral filtering technique the results can be quite consuming when it comes to resources which leads to low performance where our goal consist of less resources usage and achieving an acceptable results in high performance by doing a comparative study between the sampling distribution techniques and variance reduction techniques and different filtering techniques that are applied to the in order to enhance performance and achieve acceptable visual results.

4.3 General Design

In Figure 4.1 the general design is discussed where we split the objectives achievement into several steps where each step has an important role in providing an ambient occlusion approximation in real-time.

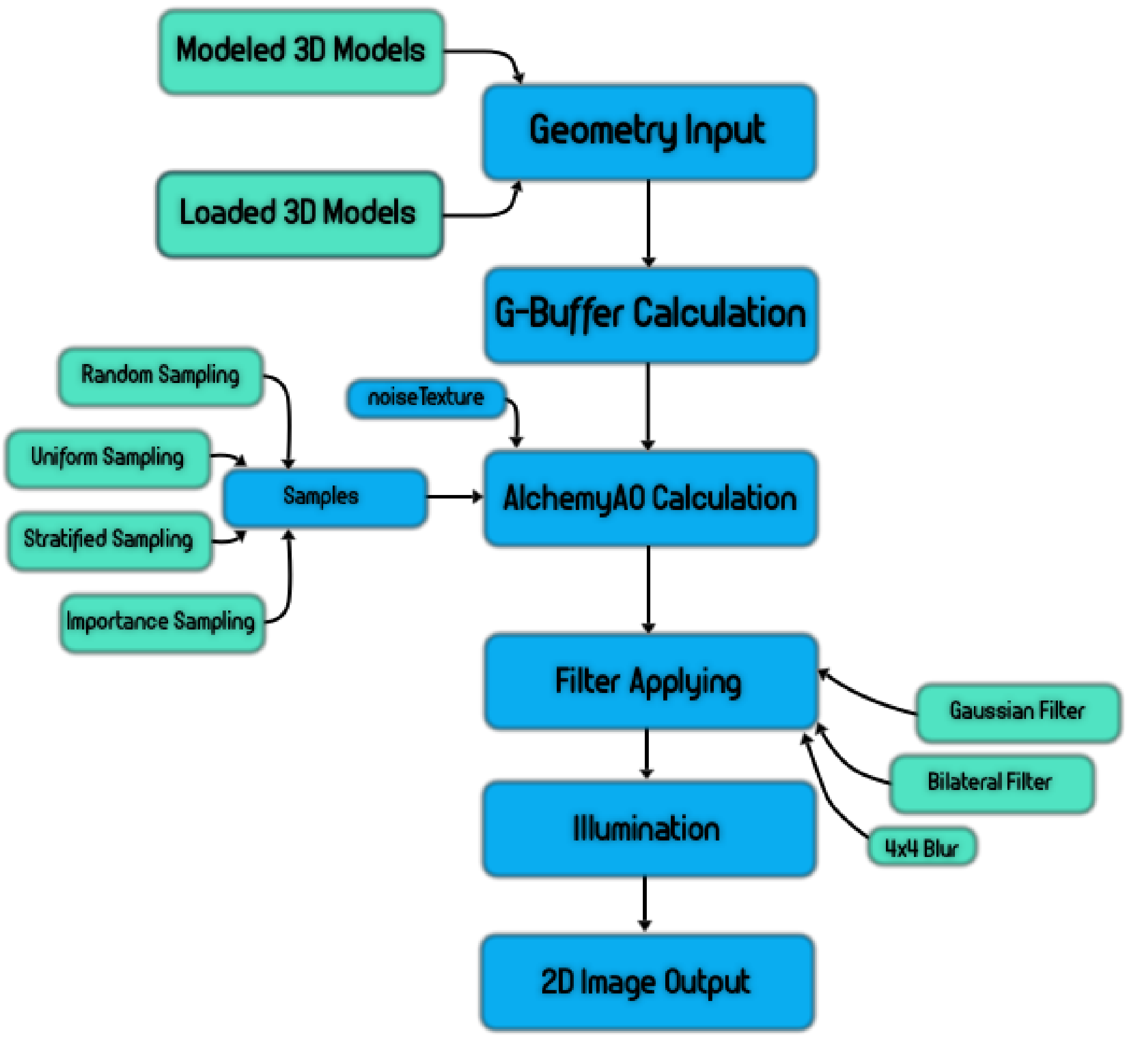


FIGURE 4.1: General Design.

4.3.1 Geometry Input

Geometry input that is used in the application it can be rather a *modeled 3D Models* which are the 3D models or objects that are modeled in execution time using the OpenGL primitive shapes (*Point, Line, Triangles, Quads, etc...*) or *loaded 3D Models* which are created using modeling softwares like *Maya* or *3ds Max* each of this loaded models is a combination of primitive shapes which is a group of polygons where a polygon is defined as a plane figure that has at least three straight sides and angles which are the shapes accepted by OpenGL that we mentioned before. Loaded models comes with an **".obj"** extension which are originally a **".txt"** files that take obj format as a known convention between modeling softwares game engines.

The obj models are usually used rather than modeling geometry shapes directly in OpenGL because of the complexity that a model can have and the high number of polygons used in order to define a complex and smooth object. In order to load this objects into the scene several libraries (APIs) are defined by experts in order to facilitate such a task, we used the Open **Ass**et **imp**ort *Assimp* Library which provides a flexible obj model loader alongside with other supported formats also it automatically generates the different textures that include geometry informations (*Normals, Specularity coefficient, Albedo*) using the **"MTL"** file that comes along with each **"OBJ"** file where it defines the placement of the textures in the concerned obj model to be used as uniform samplers in shaders. The Figure 4.2 shows an obj file which is knows with "fireplace room" that is modedled by Morgan Mcguire[51] loaded with its Albedo (Diffuse map) texture using the **Assimp** Library.

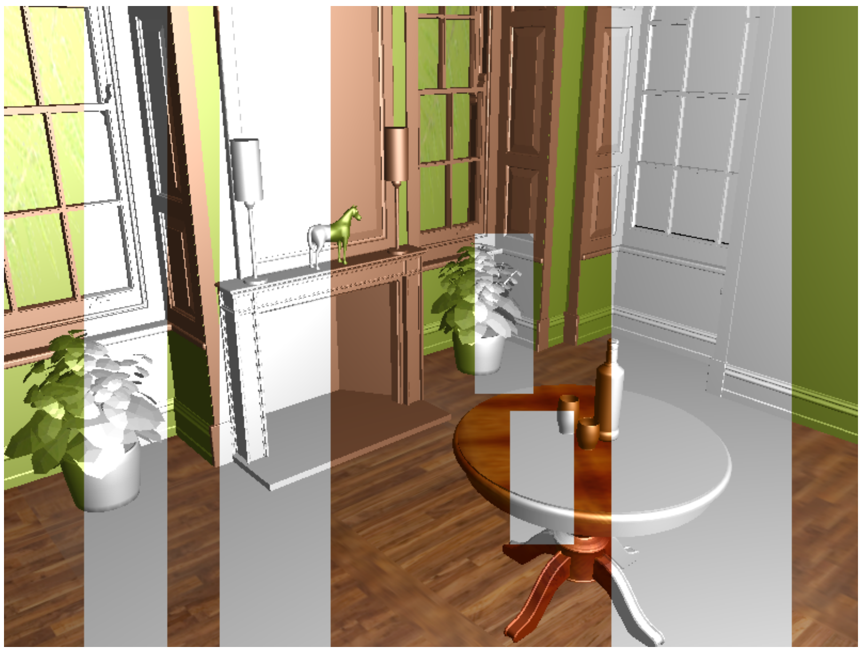


FIGURE 4.2: loaded fire place room object with and without its own textures using ***ASSIMP*** Library with FPS: 60-59 with *Blinn-Phong Local Illumination Model.*

4.3.2 G-Buffer Calculation

One of the most important things that allowed the Ambient Occlusion to be calculated in **Screen Space** is the G-Buffer where it stores all the geometry informations (*Positions*, *Normals*, *Texture Coordinates known as TexCoords*) as Figure 4.3 illustrate into a specified textures attached to the same **FBO** for later use when we use it either in approximating the Ambient Occlusion factor or to calculate the final illumination using one of the simplified local illumination models where it is considered as deferred rather than forward due to **G-Buffer** usage which is considered as a geometry pass.

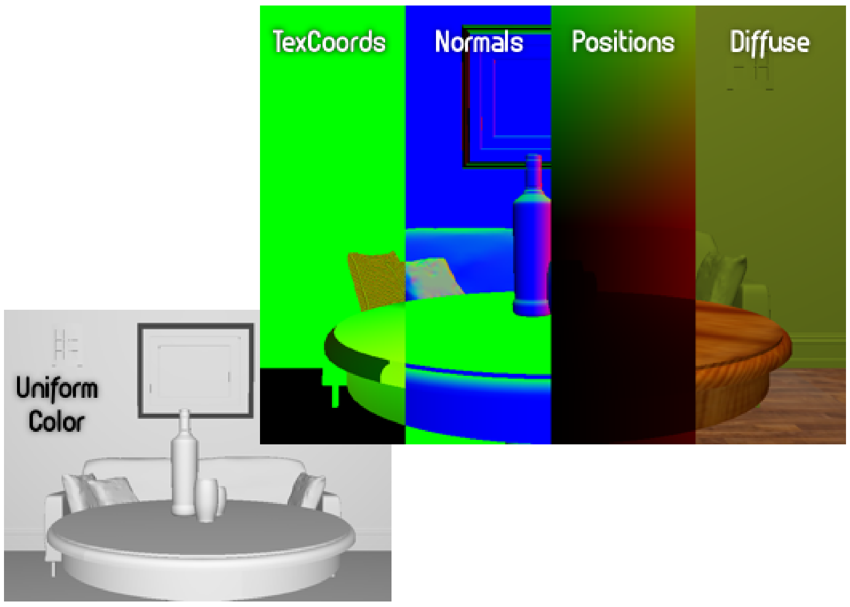


FIGURE 4.3: G-Buffer different Textures

4.3.3 Samples Creation

Approximating the Ambient Occlusion factor in screen space using the Alchemy Screen Space Ambient Occlusion (AlchemySSAO) requires kernel samples which are a limited number of points that surround a point where we want to calculate the AlchemySSAO at, where these points are generated in a way to form a hemisphere that is oriented by the surface normal which point in the Z-direction.

Generating these samples consist of a mathematical conversion between the Polar coordinates (r, θ, ϕ) and the Cartesian coordinates (x, y, z).

In a sphere with Polar coordinates (r, θ, ϕ) the Cartesian coordinates are equal to:

$$\begin{cases} x = r\sin\theta\cos\phi \\ y = r\sin\theta\sin\phi \\ z = r\cos\theta \end{cases}$$

With:

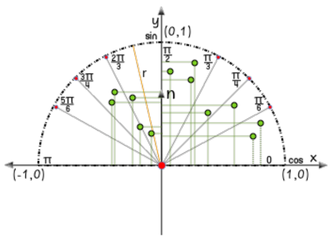
r: is the sphere radius.
To a Hemisphere variation rather than Sphere the following variation are use:

x: vary in the tangent space between $[-1, 1]$.

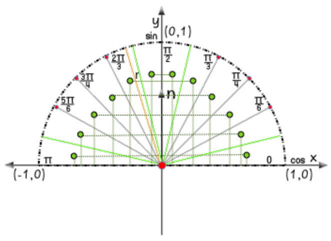
y: vary in the tanget space between $[-1, 1]$.

z: vary only between $[0, 1]$.

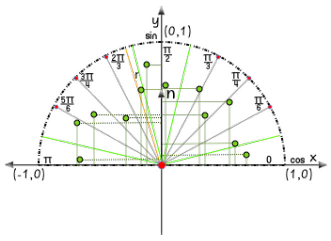
The way the kernel are generated is the essence of the AO factor approximation where we used the distribution and variance reduction techniques that are illustrated in the Figure 4.7 below:



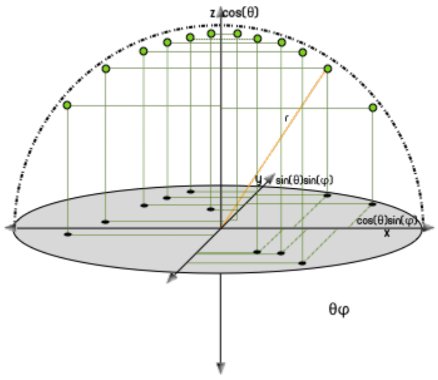
(A) Random Sampling



(B) Uniform Sampling



(C) Stratified Sampling



(D) Importance Sampling for "cos θ"

FIGURE 4.4: Used Distribution Techniques and Importance Sampling for cos θ illustration

Using only these techniques doesn't provide us with a quite great results without having a high number of samples while using a low amount of samples also result the banding effect that is noticeable to the human eye sensation that's we used a random rotation vectors to rotate samples along the z-axis these random vectors which are generated and saved into texture known as the *Noise Texture* as Figure 4.5 shows are is used to get a great results without having the banding effect which has been replaced by the noise effect that is less noticeable by human vision.

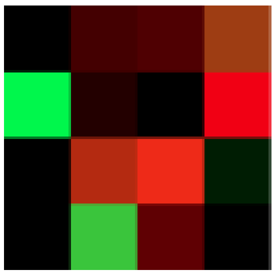


FIGURE 4.5: Noise texture used for rotating samples along z-axis

4.3.4 AlchemySSAO Calculation

After the samples are generated along side with the noise texture they are passed to the Alchemy shader in order to be rotated to produce larger variation with a low number of samples along with the geometry informations that has been stored into the gBuffer textures where these components are the essentials of approximating the AO factor in screen space using the Alchemy technique which is based on calculating the vector v_i between each sampling point which is the occluder and the center of the hemisphere that is defined as the occludee and producing a dot product between each vector and the surface normal alongside the tuning parameters which are σ as an intensity scale, k as contrast controller which is more clearly understood as the power of the AO and a depth bias product β where all this factors combined are responsible for calculating the AlchemySSAO that is saved into a texture for later display as an overdraw in a two dimensional screen resolution fit rectangle as Figure 4.6 illustrate.

The original CrytekSSAO with the hemisphere sampling improvement has been also provided alongside the AlchemySSAO to do some comparison and study the differences between the two techniques where the CrytekSSAO consist of a simple comparison between samples depth values to approximate the AO factor in the center of a hemisphere.

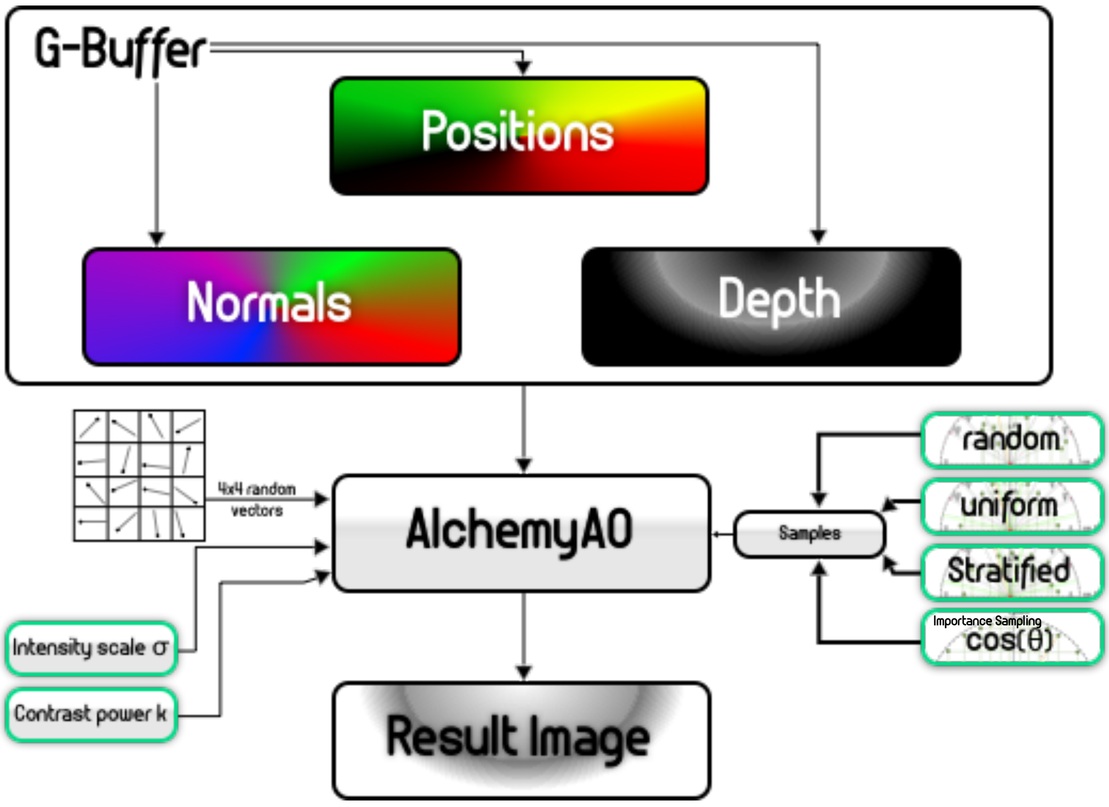
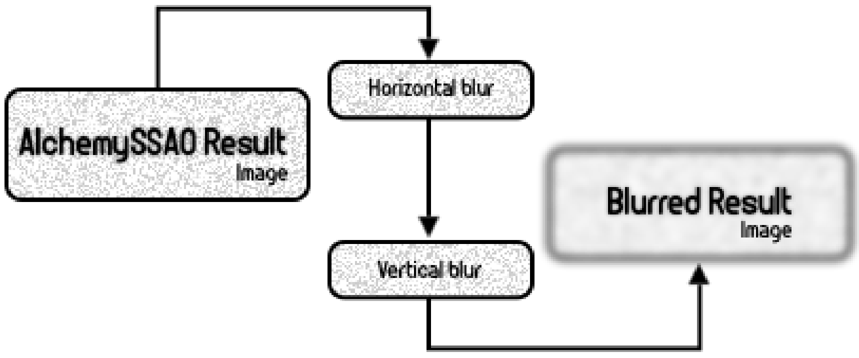


FIGURE 4.6: AlchemySSAO calculation with its essential factors illustration

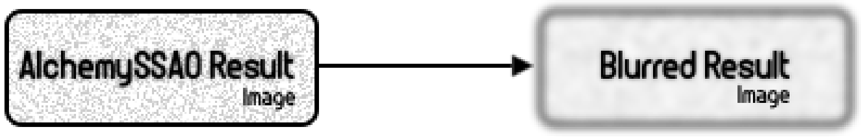
4.3.5 Filter Applying

Due the random rotation of samples along the z-axis using the random vectors as mentioned in Section 4.3.3 the banding effect which is the most observable artifact is removed and replaced with noise which is less observable to the human eye. In order to remove this kind of noise image processing techniques are needed specifically filtering which is known as "*blurring*" nowadays which is the right solution to achieve such a task.

Three types of filters has been implemented which are the original AlchemySSAO used filter that takes an noisy image as input and result a blurred image as in Figure 4.7(b) that is defined as the *bilateral* filter which is a powerful filtering technique in edge preserving that uses a range parameter σ_r and spatial parameter σ_s originally based on the second filter which is the *gaussian* filter that split the blurring phase into two phases the horizontal blurring and the vertical blurring as illustrated in Figure 4.7(a) which is preferred in order to augment performance by averaging a pixel color using pixels that correspond to the blur size parameter with a the targeted pixel in the center and a blur strength parameter for smoothing while the last filter is a 4x4 grid simple blur filter that average a 4x4 texels which is the same resolution of the noise texture in order to provide a simple blur efectFigure 4.7(b).



(A) Gaussian blurring technique phases.



(B) Bilateral blurring and 4x4 blurring filters phase.

FIGURE 4.7: Different Blurring techniques which are used in noise suppression illustration.

4.3.6 Illumination

Approximating the ambient light alone doesn't achieve realistic scene rendering due the fact that real-world scenes doesn't only includes the ambient light for that kind of reason we used a local illumination model in order to enhance the realism of the scene which is the *Blinn-Phong* model that is inherited from the *phong* where it needs less calculation than the Phong model due to the use of the halfway vector rather than the reflective vector which requires less computations that is resulted from using only the light and view vectors also it also enhance specularity realism by blending the specular component a little bit with the model matriel itself.

4.4 Realization

In order to achieve our goal a graphical application is needed to render or rasterize in more accurate way a virtual scene with an AlchemySSAO factor and a simplified local illumination model where we used a set of softwares and materials alongside the application structure which are presented in the next sections.

Also some softwares are used in order to design the schemes that are used in this thesis which are also presented in the section below.

4.4.1 Used Softwares and Materials

In general an application implementation and design requires softwares and materials full specification where it comes handy for future comparisons or to see how much of achievement has been accomplished by a specific materials and softwares. In this part we provided the full specification of materials, softwares along side the application structure.

4.4.1.1 Materials

⇒ Machine: MacBook pro Mid 2010 Series .

⇒ Operation System: **macOS** High Sierra, Version: 10.13.3 .

⇒ Processor: 2.4 GHz Intel Core 2 Duo .

⇒ Memory: 6 GB 1067 MHz DDR3 .

⇒ Graphic card: NVIDIA GeForce 320M 256 MB .

4.4.1.2 Softwares and APIs

Several Softwares and APIs are used in order to accomplish the application and the figures and schemes used in this thesis where they are introduced below.

4.4.1.2.1 Softwares

⇒ Programming Environment: Apple Xcode, Version 9.2(9C40b) .

⇒ Figures design: Adobe Photoshop CC, Version 2017.1.1 .

⇒ Schemes design: Sketch, Version 43.1 .

⇒ Metrics (RMSE, Visual): Matlab, Version R2015b .

4.4.1.2.2 APIs

⇒ **G**raphics **L**ibrary: Open**GL**, profile: Core, Version 3.30 .

⇒ Open**GL** **S**hading **L**anguage: GLSL, Profile: Core, Version 3.30.

⇒ Open**GL** **M**athematics: GLM, Core profile : Supported, Version 0.9.9.0 .

⇒ 3D Models Loader: Open **Ass**et **Imp**ort Library (Assimp), Version 4.1.0 .

⇒ Loader Generator for C/C++: GLAD, Version: 0.1.13a0 .

⇒ Windowing System and inputs: GLFW, Version 3.2.

⇒ Text Render: FreeType, Version 2.8.1 .

⇒ Image Save: FreeImage, Version 3.17.0.

4.4.2 Application Structure

In order to calculate the AlchemySSAO and using blurring techniques we used shaders that are necessary for the total flexibility and freedom in rendering that make the over-drawing possible where the task consist of using 5 rendering passes where 4 are used for AlchemySSAO and blurring calculation and additional shader are used to render textual content to the scene which contains informations about used parameters and material also the fps and the ms per frame where the list below consist of detailed informations about each pass.

⇒ First shader: G-Buffer setting in order to obtain the geometry informations that is stored into defined textures (Positions, Normals, Albedo, Depth) that are attached to the gbuffer fbo.

⇒ Second shader: AlchemySSAO factor calculation using the gbuffer textures, chosen samples, random vectors for rotations.

⇒ Third shader: burring resulted image from the previous shader using bilateral, gaussian or 4x4 blur filter.

⇒ Fourth shader: Local illumination calculation with the Blinn-Phong model.

⇒ Additional shader: rendering the textual content into the scene.

4.4.2.1 First Shader: G-Buffer

Using Screen Space to approximate the AO factor requires using textures which is a known fact where they are actually a set images that contain the geometry informations stored using the gbuffer shader by creating a framebuffer that holds these different needed textures: Positions, Normals, Albedo, Depth, where these informations are obtained from the loaded 3D model.

Positions, Normals, are provided by the VAO to the vertex shader via the attributes arrays pointers to send the values from the cpu directly to the vertex shader where the normals are multiplied by the Normal Matrix for the correct bent and the position are multiplied by the view and model matrices in order to extract the depth value which is linear in the view position case directly from the position after that each position is multiplied by the projection matrix in order to be displayed according to the projection matrix. After that each of the normals and positions are passed to the Fragment Shader as an out variable.

Additionally the Albedo of a certain model are recovered in the fragment shader using the sample2D uniform same as the specular component after that we normalize the normals, now that we have the three needed components we attach their data according to their color attachment location index using the layout keyword and specifying the out variable for each location where these out variables are containing the geometry informations which are stored into textures after a draw call to the used color attachements

4.4.2.2 Second Shader: AlchemySSAO Calculation

In order to calculate the AlchemySSAO factor several steps are needed within the AlchemySSAO shader itself where they are explained below.

4.4.2.2.1 Display Screen Subdivision

After sending the 4x4 noise texture to the AlchemySSAO fragment shader as a uniform sampler2D along with other uniforms which are used to tune the effect a 2D vector is required in order to scale the noise texture all over the screen resolution where its x and y values equals to window width and height by 4 division resulting a scaled 4x4 texture that match the window resolution.

4.4.2.2.2 Sampling Core

Now that the random vector are distributed along the viewing screen a sample core is transferred from the cpu to the fragment shader as a uniform vector of (x,y,z) vectors which contains a set of samples according the needed number of samples which is known as the *Kernel Size* generated by one of the fourth used techniques: random, uniform, stratified distribution techniques, or an importance sampling variance reduction technique according to the $\cos(\theta)$ function where the resulted samples are distributed along a hemisphere with a radius r.

4.4.2.2.3 AlchemySSAO factor Calculation

Now we come to the interesting part which is the factor calculation after we provide the essentials needs, each of the positions and normals along with the random vectors are obtained from textures where we create a TBN matrix that switch between spaces (*tangent space*, *view space*) using the normal vector and the random vector to create th tangent and bitangent vectors resulting each sample transformation from the tangent space to the view space along the surface normal where we translate each sample according the first viewed position that is presented in the positions texture.

After that each of these samples are projected into the screen to get their positions on texture where these samples are used as an offset in order to get the occludee position that is involved in the calculation of v_i vector alongside with the occluder position which is the sample position itself.

The vector v_i along with other tuning parameters are used then to calculate the AlchemySSAO factor using the monte carlo estimator that is defined by the original authors of the AlchemySSAO technique.

4.4.2.3 Third Shader: Filtering

AlchemySSAO results that is stored in a texture contains an amount of noise that is produced from low sampling core samples and the usage of the rotation vectors that's why the resulted texture (image) is passed to a filtering or in more accurate way blurring shader where it contains three types of blurring techniques which are mentioned in Section 4.3.5.

The first is the gaussian blur which uses two horizontal and vertical blur passes for performance reasons using a kernel size which indicates the grid involved in blurring averaging and a strength value to define more or less blur effect while we offset each texture coordinate by the exact size of a single texel multiplied by either a horizontal-vertical blurring stage 0-1 values respectively to calculate the average value that is divided by a coefficient sum which contains an incremental gaussian coefficient values.

The second one is a simple 4x4 blur stage where we used a 4x4 grid to average 4x4 texels using an offset vector to offset the resulted noisy image by the exact size of a single texel in order to provide a simple blurring results.

Lastly the bilateral filter which is based on the gaussian filter where we used two nested loops along with a range and spatial σ values which make the bilateral filter a geometry-aware filter when in comes to pixel (texels) edges preserving where an offset color is chosen to present one of the neighbors AlchemySSAO factor value using a 2D vector for the current position added to the actual texture coordinate and divided by the actual texture size that is presented using a 2D vector, after that the spatial distance is computed by the distance between the centered targeted texel and one of its neighbors and the range distance which is calculated using the offset color of a neighbor texel value and the actual AlchemySSAO factor value in the current position, where these two distances are used to calculated the weight assigned to a neighbor texel in order to blur the current texel after normalizing each weight.

4.4.2.4 Fourth Shader: Illumination

we compute the illumination using the Blinn-Phong local illumination model that we came through in Chapter 1, Section 2.2.1.2.2 with an extended ambient light component in order to overdraw our resulted AlchemySSAO factor over the geometry and that is achieved by either replacing the uniform ambient light factor by the AlchemySSAO factor or by multiplying it directly by approximated AO value alongside with the diffuse, specular components, each of the scene materials can be either a uniform colorization or a used Albedo that is multiplied by the Alchemy factor to reduce the high intensity of colors which reflect the real-world when it comes to the ambient component and multiplying the albedo map directly with the light intensity I_d and the Lambert model $\cos(\theta)$ for the diffuse component while the specular component is calculated using halfway and view vectors.

The Blinn-Phong formula after including the approximated AO factor:

$$I = AlchemySSAO * K_a + I_d * K_d * (\hat{N} \cdot \hat{L}) + I_s * K_d * (\hat{H} \cdot \hat{N})^n$$

4.4.2.5 Additional Shader: Text render

In order to display textual content into the screen frame an additional shader is used which takes as an input a uniform texture where its Red component contains an orthogonal projected characters where each character is within a small quad that is rendered into the Alpha channel in order to have a transparent background and only the actual character pixels rendered and that is achieved by enabling the blending OpenGL technique and using the FreeType Library.

The Figure 4.8 defines the global design of the graphical application with the different typeso f communications between the three components: CPU, Shaders, GPU.

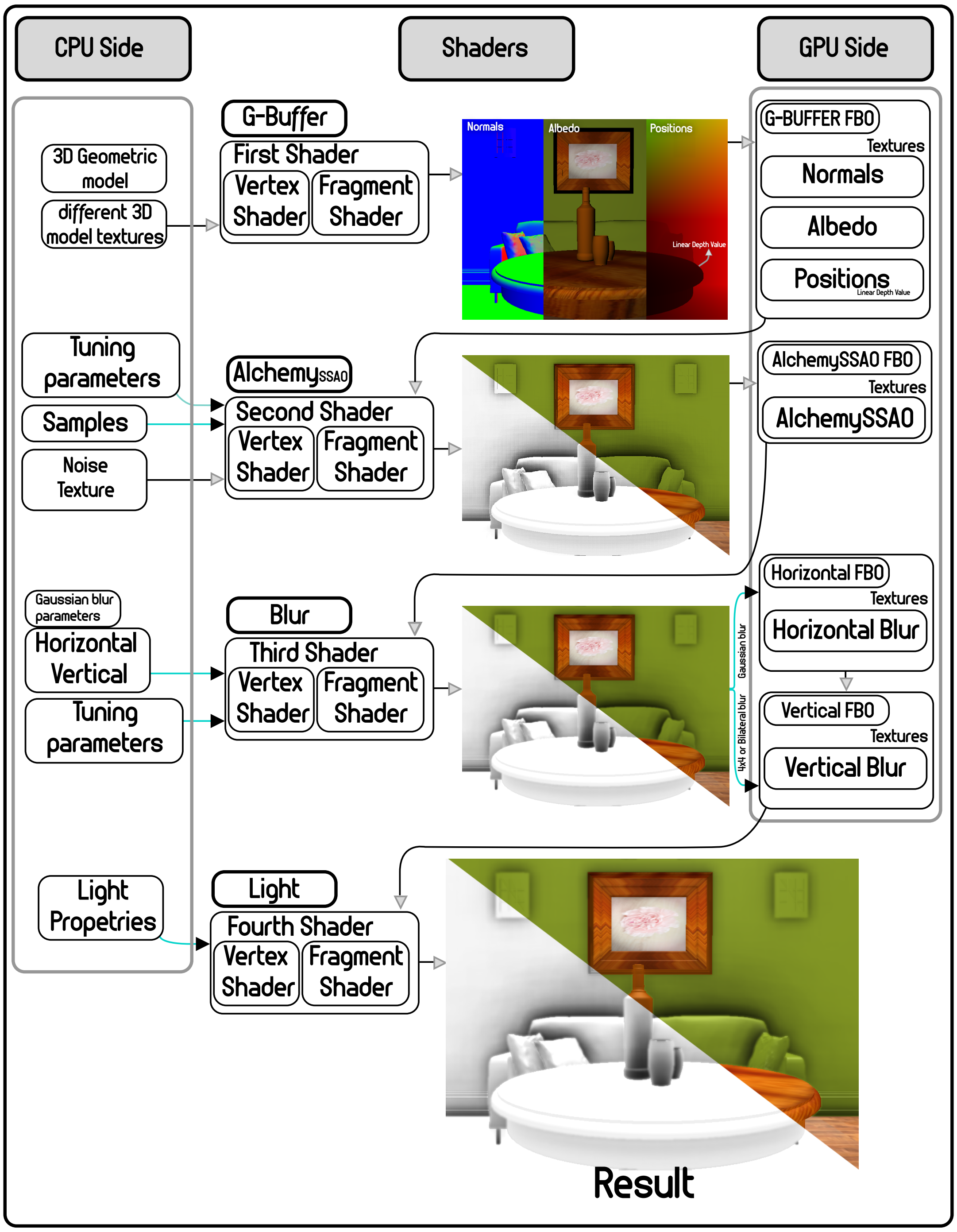
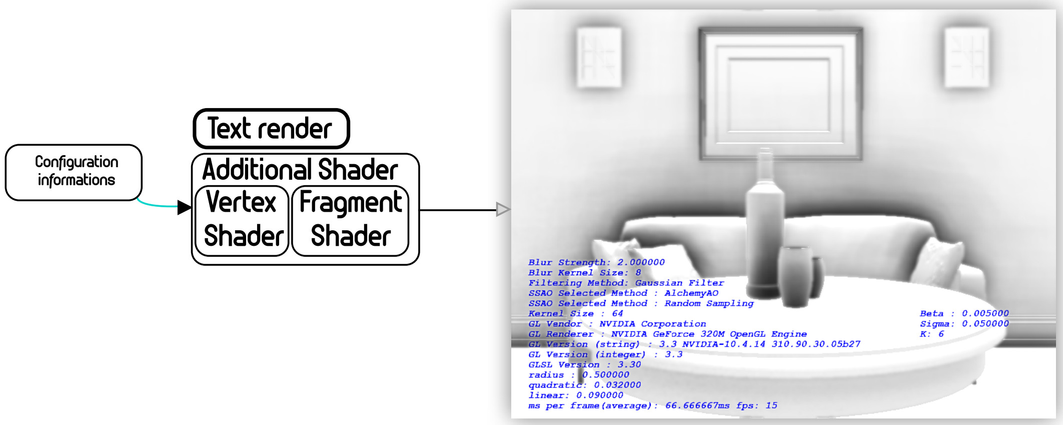


FIGURE 4.8: Global structure of the graphical application.

The structure of the text render shader are also provided in the Figure 4.9 below.



```
Blur Strength: 2.000000
Blur Kernel Size: 8
Filtering Method: Gaussian Filter
SSAO Selected Method : AlchemyAO
SSAO Selected Method : Random Sampling
Kernel Size : 64                                              Beta : 0.005000
GL Vendor : NVIDIA Corporation                                Sigma: 0.050000
GL Renderer : NVIDIA GeForce 320M OpenGL Engine               K: 6
GL Version (string) : 3.3 NVIDIA-10.4.14 310.90.30.05b27
GL Version (integer) : 3.3
GLSL Version : 3.30
radius : 0.500000
quadratic: 0.032000
linear: 0.090000
ms per frame(average): 66.666667ms fps: 15
```

FIGURE 4.9: Text render shader additional shader structure.

4.5 Results

The Original AlchemySSAO technique has been implemented which uses both random samples distribution technique and bilateral filtering but using randomness and bilateral filter causes ignoring an important depth values occasionally that needs to be sampled which requires a high kernel size leading to low performance which also caused by the usage of the bilateral filter. In this part we tried to enhance the usage of samples by doing a comparative study between the different distribution techniques and also using an importance sampling variance reduction technique by choosing $\cos\theta$ as a pdf to the original AlchemySSAO monte carlo estimator for samples weighting in order to provide an acceptable results comparing to a reference image that is rendered using 576 samples in five different scenes using fixed radius and tuning parameters along filtering parameters and from the other hand the same comparison are done except rather than using fixed parameters this time we control the different used parameters in order to get much better results like hemisphere sampling radius, tuning parameters, also replacing the bilateral filter by either a gaussian filter or 4x4 simple blur can increase the performance significantly and also provide an acceptable results.

4.5.1 Metrics

In order to achieve the comparison between the different samples techniques images and the reference image we used the following different metrics:

$\Rightarrow$ Visual Metric: which is defined as the difference between the reference image and the different resulted images where the difference is calculated by subtracting the R,G,B values between reference image pixels and resulted image pixels

⇒ RMSE Metric: the root mean squared error which is also known as the *root mean squared deviation* is one another way to calculated the amount of difference between two images more accurately by using the following formula:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(I_{ref,i} - I_{res,i})^2}{n}}$$

With:
n: the resolution of the images.
i_{ref}: is the reference image.
i_{res}: is the resulted image.

4.5.2 G-Buffer Implementation

AlchemyAOcalculation In the Figure 4.13 the G-Buffer implementation are illustrated where it contains the different geometry informations (Positions, Normals, Albedo) the depth buffer isn't included due the fact that we extract the linear depth value from the position information while it is in the view space where it is known that the depth is linear before projecting positions into the screen where the depth becomes non-linear.

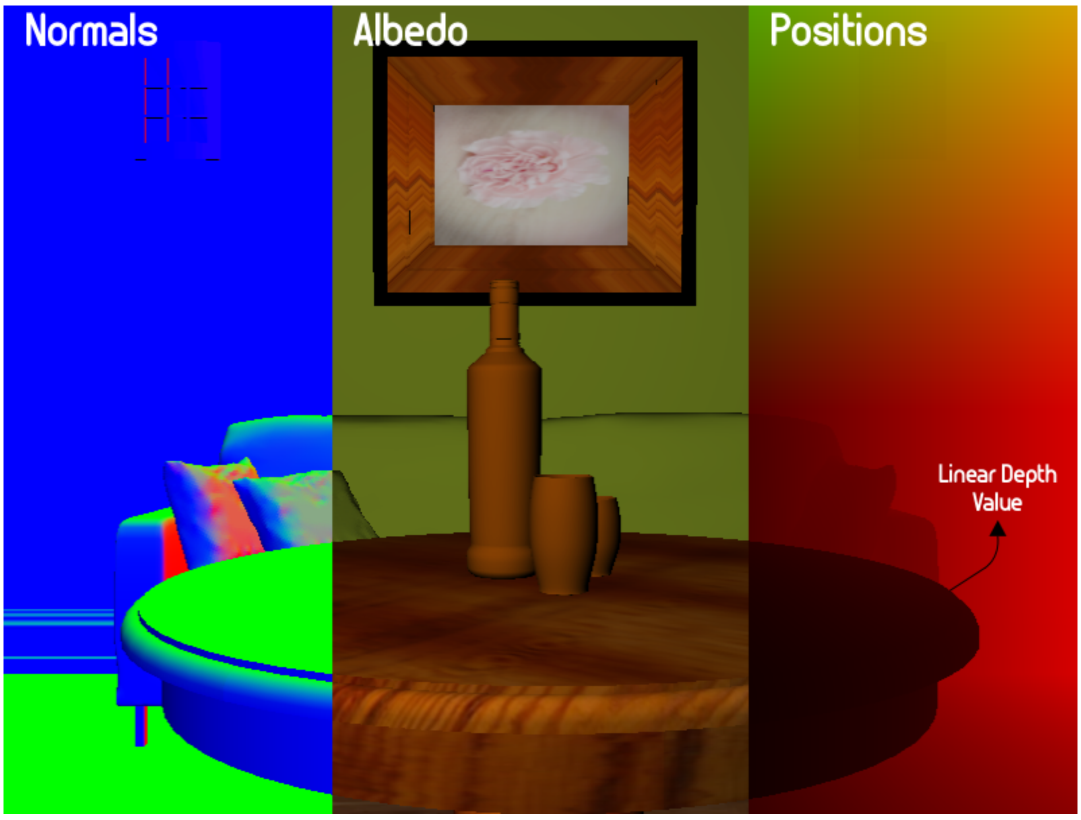


FIGURE 4.10: G-Buffer implementation results.

4.5.3 AlchemySSAO Results

The AlchemySSAO factor are calculated as it is shown in Figure 4.11 using the fireplace room 3D model provided by Mcguire[51] where the illustration is given with and without texture using the following parameters:

AlchemySSAO parameters:

⇒ radius : 0.1.

⇒ $\beta = 0.005$.

⇒ $\sigma = 0.05$.

⇒ Contrast power : $k = 6$.

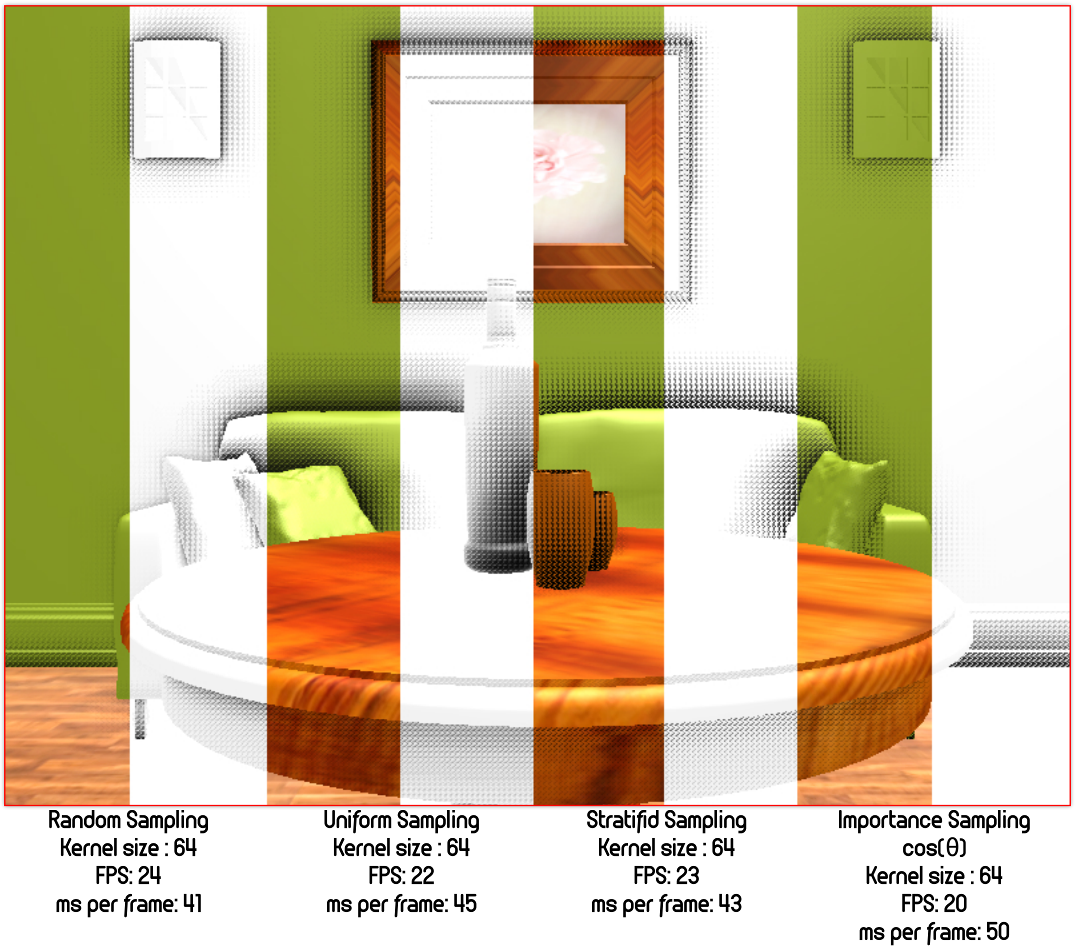


FIGURE 4.11: AlchemySSAO factor calculated using random, uniform, stratified, importance sampling techniques.

As we can see the resulted AlchemySSAO output depends on the used sampling techniques and the complexity of geometry which gives different results with a fps variation.

Compared to the improved CrytekSSAO technique:
CrytekSSAO parameters:

⇒ radius : 0.1.

⇒ SSAO power : 3.

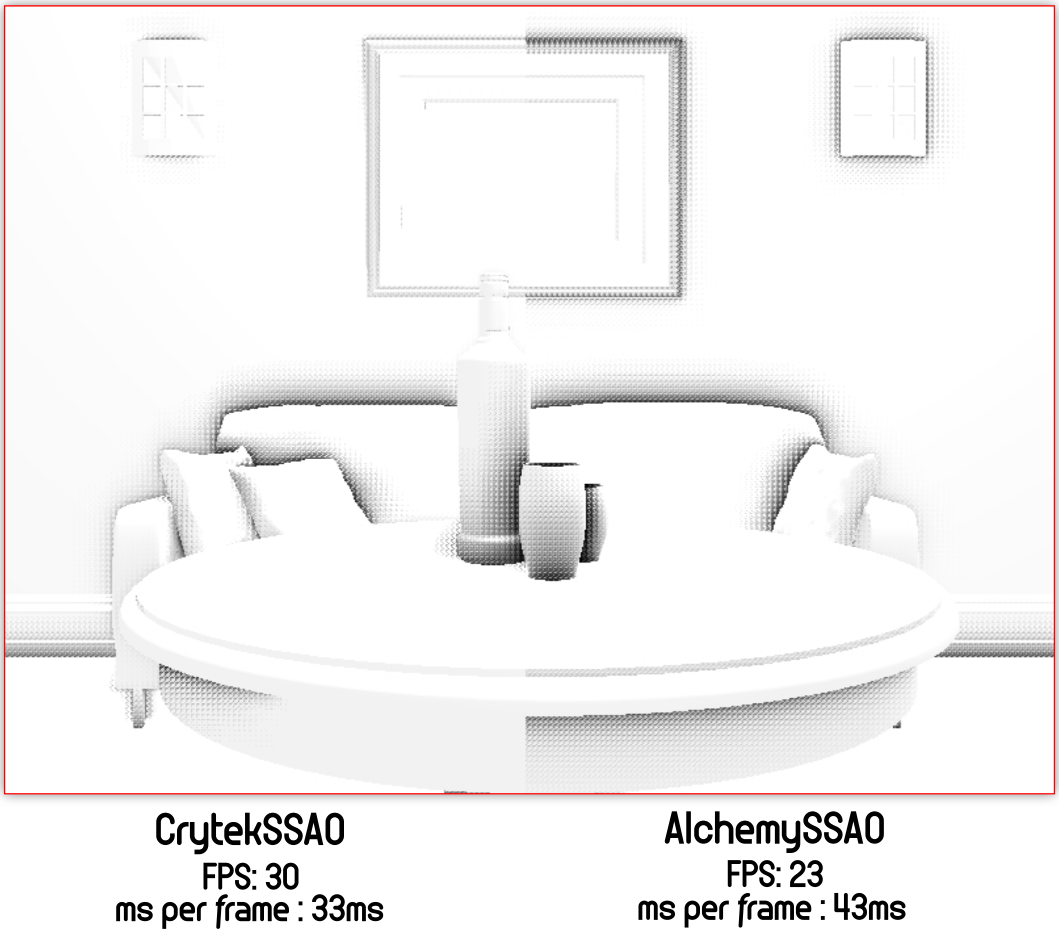


FIGURE 4.12: Comparison between the CrytekSSAO and the AlchemySSAO techniques for ambient occlusion approximation.

The result above clearly illustrate the efficiency of the AlchemySSAO compared to the CrytekSSAO technique though when it comes to performance the CrytekSSAO considered faster than the AlchemySSAO.

4.5.4 Sampling Core

Sampling Core play an essential role in enhancing the AlchemySSAO factor due the fact that the more we samples the more we get greater result and the opposite that's why is this part we compare different samples kernel size using random distribution and the fireplace room 3D model for the illustration which is provided from[51] as Figure 4.13 illustrates using the following parameters:

AlchemySSAO parameters:

⇒ radius : 0.5.

⇒ $\beta = 0.005$.

⇒ $\sigma = 0.05$.

⇒ Contrast power : $k = 6$.

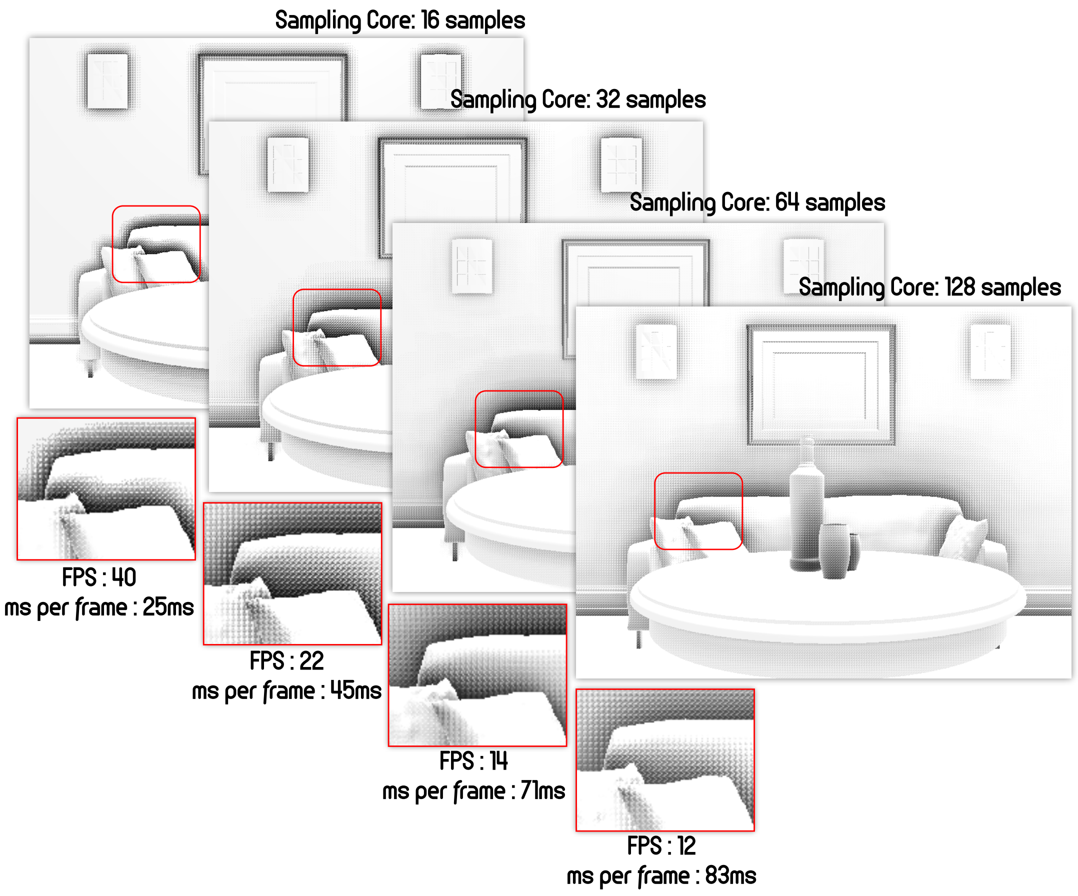


FIGURE 4.13: enhanced results due the sampling core size variation.

4.5.5 Sampling Distribution Techniques

In order to variate samples distribution different distribution techniques are implemented which are the random, uniform and stratified techniques along with an importance sampling according to the pdf $\frac{\cos(\theta)}{\pi}$.

Visual and RMSE metrics are computed in order to compare between the different techniques mentioned above for 5 different scene which are:

⇒ Sponza[51].

⇒ Living room[51].

⇒ fireplace room[51].

⇒ Sibenik[51].

⇒ Dragon.

With the following AlchemySSAO factor and Filtering parameters:
AlchemySSAO parameters:

⇒ radius : 0.5.

⇒ $\beta = 0.005$.

⇒ $\sigma = 0.05$.

⇒ Contrast power $k = 6$.

Filtering parameters:

⇒ Filter: gaussian

⇒ Blur strength = 2.0.

⇒ Blur Kernel Size = 8.

Where the AlchemySSAO tuning parameters and hemisphere radius along filtering parameters are considered the same to all tested scenes to compare them exactly with the same way.

Sponza:

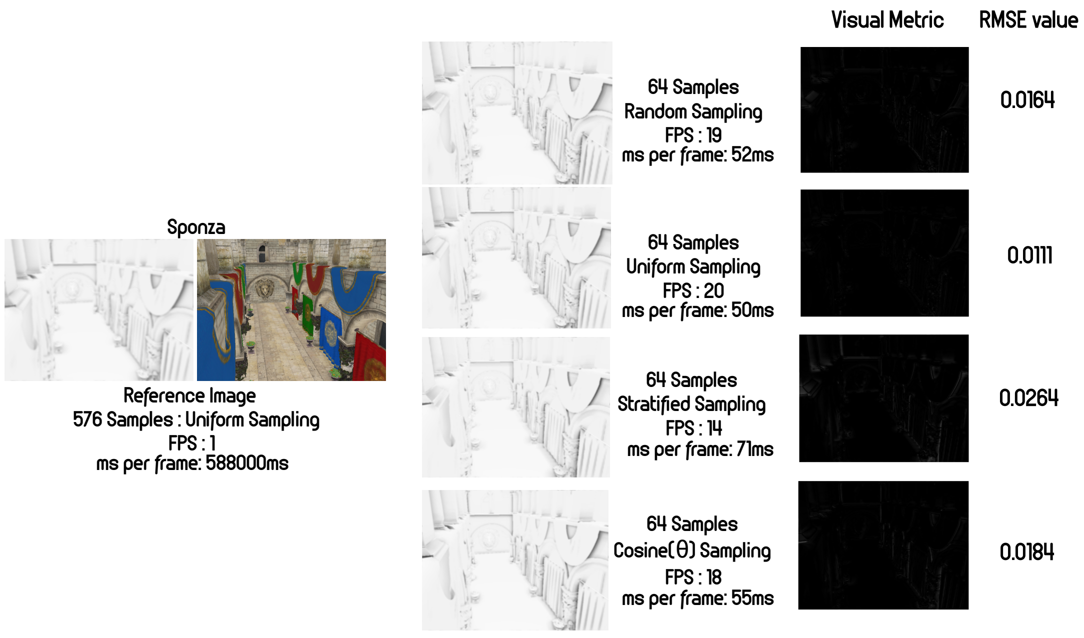


FIGURE 4.14: Comparison of Sponza 3D model between the reference image and implemented techniques using visual and root mean squared error metrics.

fireplace room:

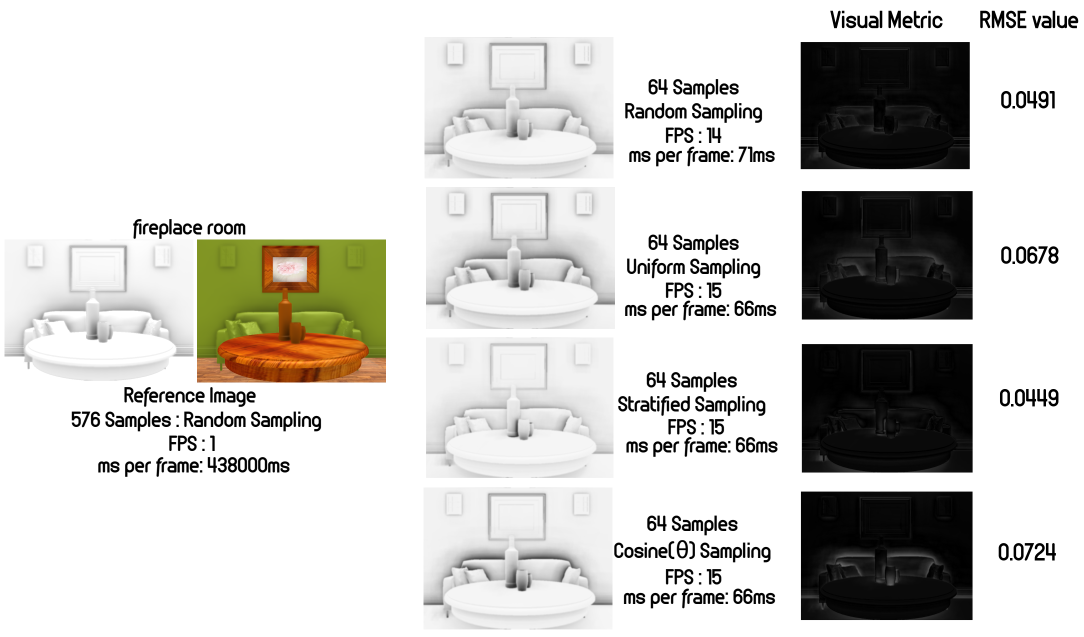


FIGURE 4.15: Comparison of Fireplace room 3D model between the reference image and implemented techniques using visual and root mean squared error metrics.

Sibenik:

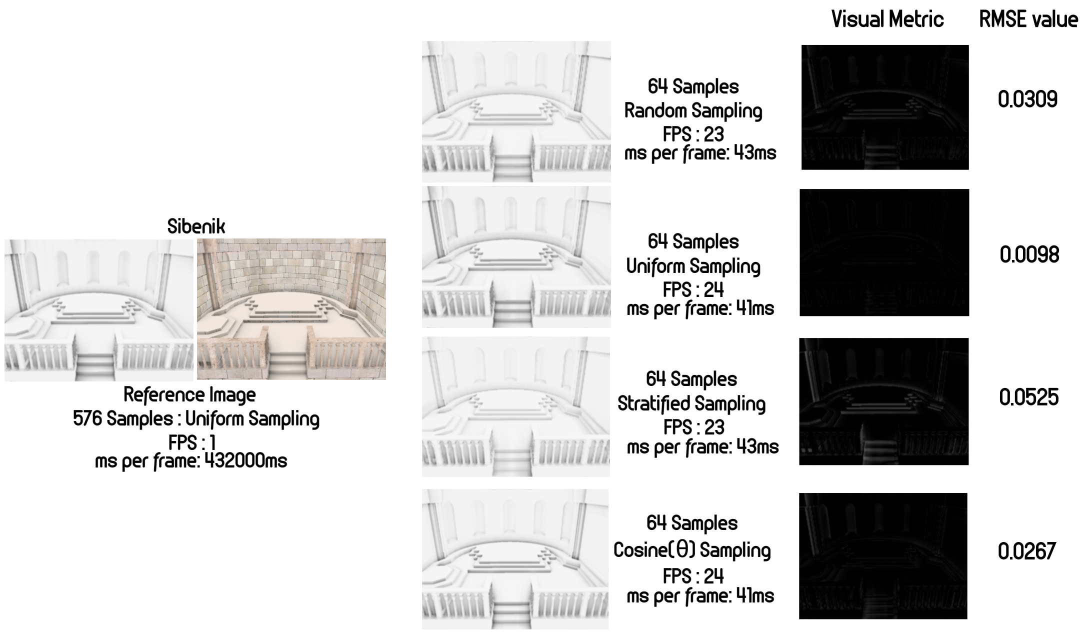


FIGURE 4.16: Comparison of Sibenik 3D model between the reference image and implemented techniques using visual and root mean squared error metrics.

Living room:

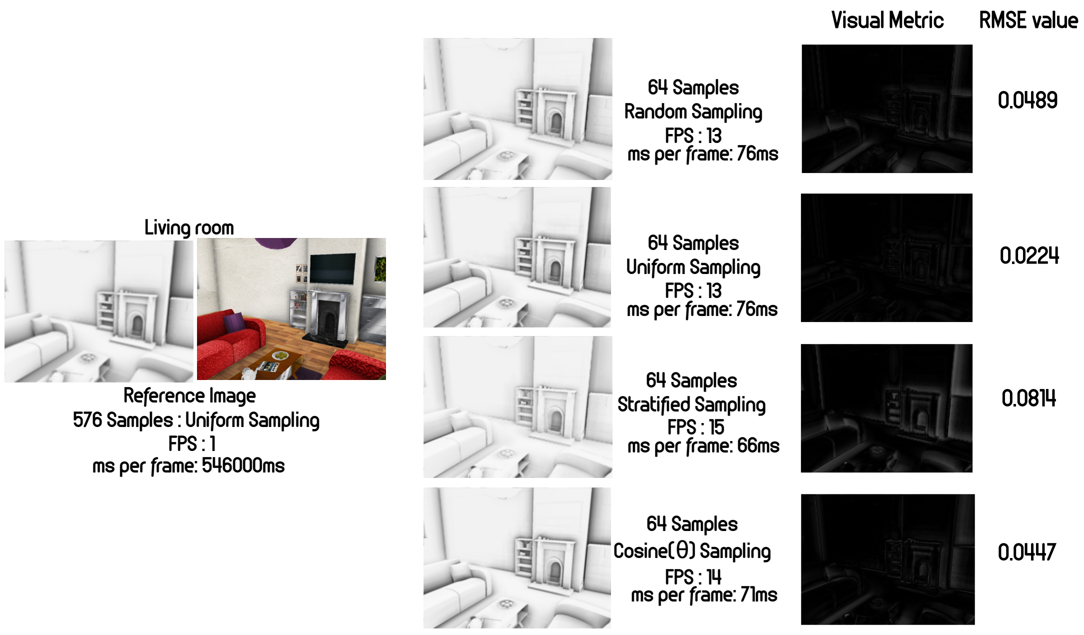


FIGURE 4.17: Comparison of Living room 3D model between the reference image and implemented techniques using visual and root mean squared error metrics.

Dragon:

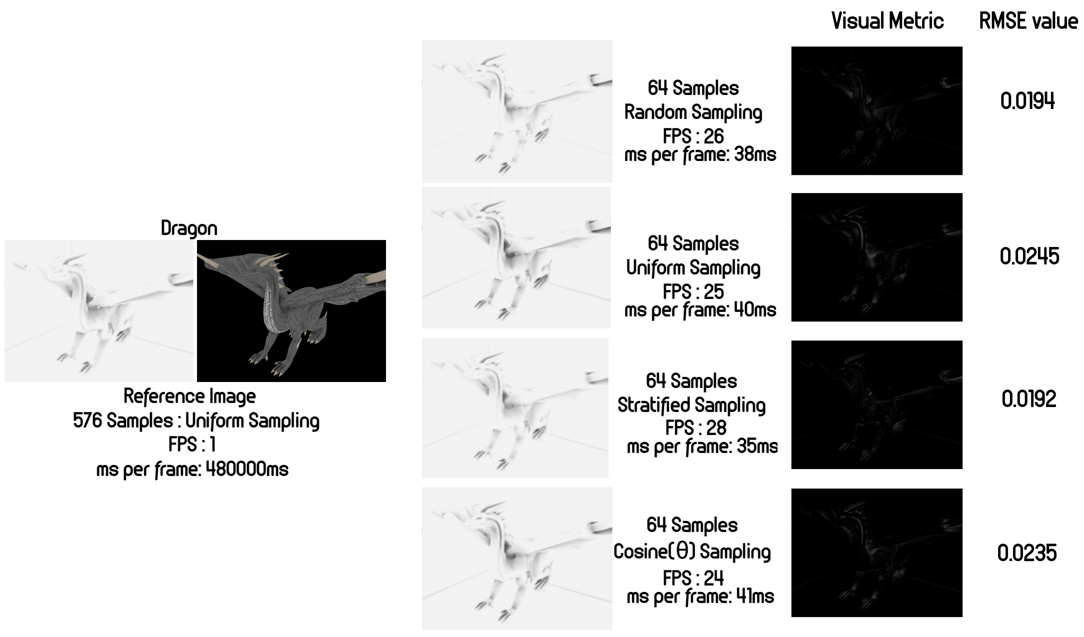


FIGURE 4.18: Comparison of Dragon 3D model between the reference image and implemented techniques using visual and root mean squared error metrics.

From the comparisons above we see that each of the used sampling techniques along filtering can effect the resulted output which is considered as an acceptable result comparing to the reference image in each tested scene due the fact that both hemisphere radius and tuning parameters are fixed.

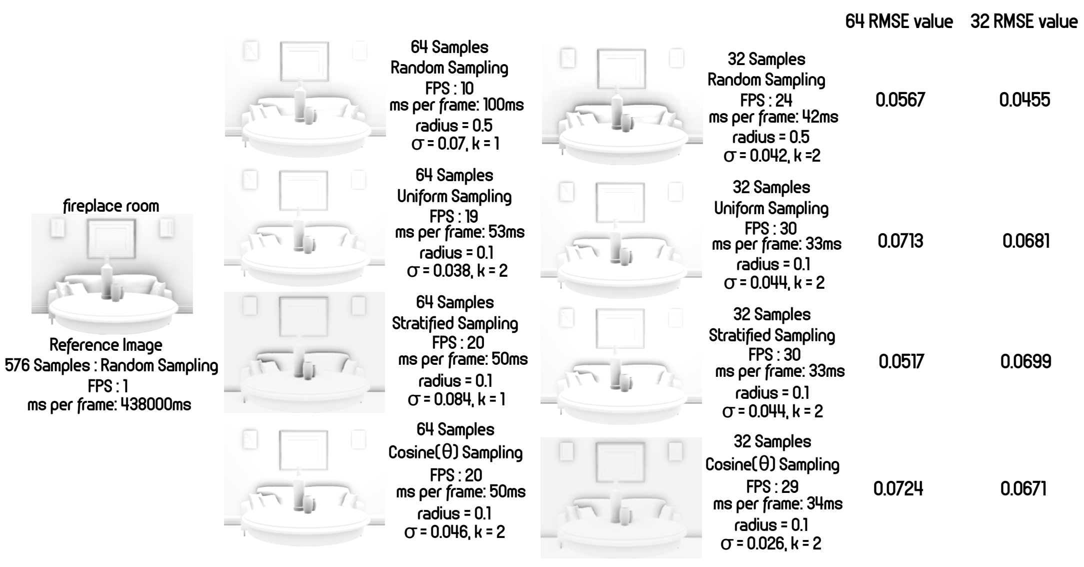
After that we do a comparison between 3 chosen scenes with different tuning parameters which are the intensity scale σ and the contrast power k and sampling hemisphere radius r in order to get much more acceptable results with higher performance and small difference while the chosen scenes are :

⇒ fireplace room[51].

⇒ Sibenik[51].

⇒ Living room[51].

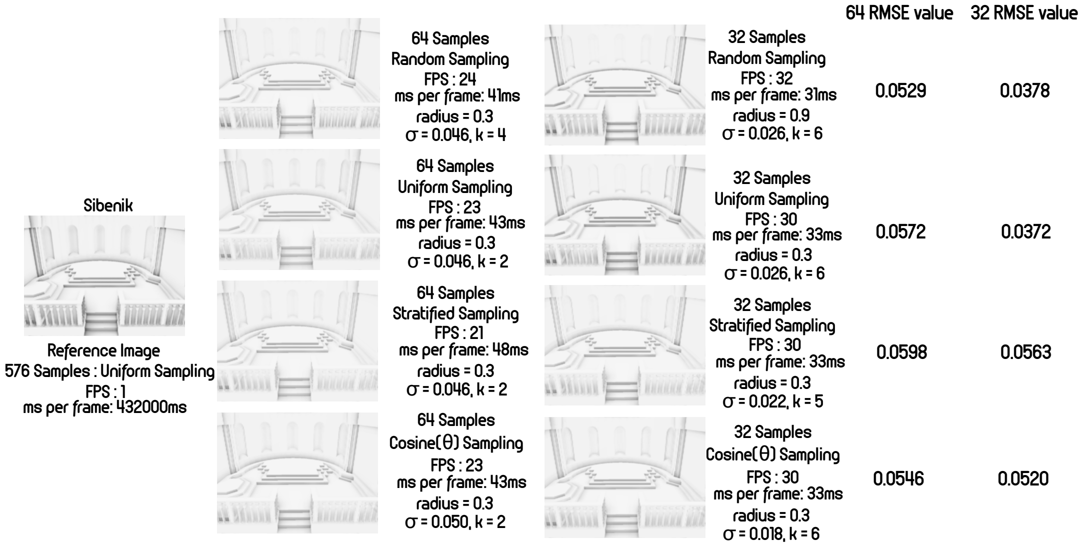
fireplace room:



		64 RMSE value	32 RMSE value
64 Samples Random Sampling FPS : 10 ms per frame: 100ms radius = 0.5 σ = 0.07, k = 1	32 Samples Random Sampling FPS : 24 ms per frame: 42ms radius = 0.5 σ = 0.042, k =2	0.0567	0.0455
64 Samples Uniform Sampling FPS : 19 ms per frame: 53ms radius = 0.1 σ = 0.038, k = 2	32 Samples Uniform Sampling FPS : 30 ms per frame: 33ms radius = 0.1 σ = 0.044, k = 2	0.0713	0.0681
64 Samples Stratified Sampling FPS : 20 ms per frame: 50ms radius = 0.1 σ = 0.084, k = 1	32 Samples Stratified Sampling FPS : 30 ms per frame: 33ms radius = 0.1 σ = 0.044, k = 2	0.0517	0.0699
64 Samples Cosine(θ) Sampling FPS : 20 ms per frame: 50ms radius = 0.1 σ = 0.046, k = 2	32 Samples Cosine(θ) Sampling FPS : 29 ms per frame: 34ms radius = 0.1 σ = 0.026, k = 2	0.0724	0.0671

FIGURE 4.19: Comparison between a set of resulted images from rendering fireplace room with different radius and tuning parameters.

Sibenik:



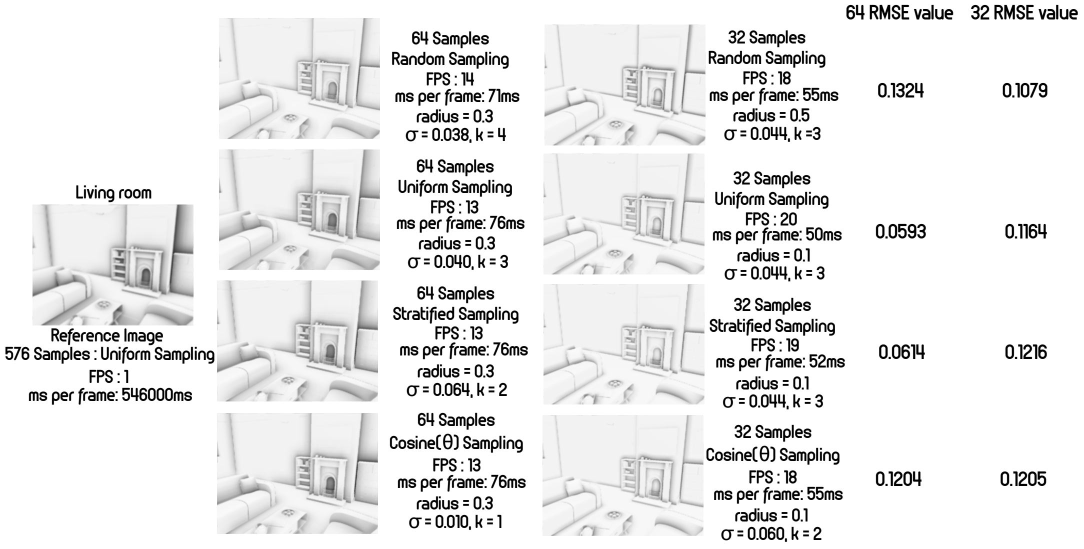
Sibenik

Reference Image
576 Samples : Uniform Sampling
FPS : 1
ms per frame: 432000ms

		64 RMSE value	32 RMSE value
64 Samples Random Sampling FPS : 24 ms per frame: 41ms radius = 0.3 σ = 0.046, k = 4	32 Samples Random Sampling FPS : 32 ms per frame: 31ms radius = 0.9 σ = 0.026, k = 6	0.0529	0.0378
64 Samples Uniform Sampling FPS : 23 ms per frame: 43ms radius = 0.3 σ = 0.046, k = 2	32 Samples Uniform Sampling FPS : 30 ms per frame: 33ms radius = 0.3 σ = 0.026, k = 6	0.0572	0.0372
64 Samples Stratified Sampling FPS : 21 ms per frame: 48ms radius = 0.3 σ = 0.046, k = 2	32 Samples Stratified Sampling FPS : 30 ms per frame: 33ms radius = 0.3 σ = 0.022, k = 5	0.0598	0.0563
64 Samples Cosine(θ) Sampling FPS : 23 ms per frame: 43ms radius = 0.3 σ = 0.050, k = 2	32 Samples Cosine(θ) Sampling FPS : 30 ms per frame: 33ms radius = 0.3 σ = 0.018, k = 6	0.0546	0.0520

FIGURE 4.20: Comparison between a set of resulted images from rendering sibenik with different radius and tuning parameters.

Living room:



Living room

Reference Image
576 Samples : Uniform Sampling
FPS : 1
ms per frame: 546000ms

		64 RMSE value	32 RMSE value
64 Samples Random Sampling FPS : 14 ms per frame: 71ms radius = 0.3 σ = 0.038, k = 4	32 Samples Random Sampling FPS : 18 ms per frame: 55ms radius = 0.5 σ = 0.044, k =3	0.1324	0.1079
64 Samples Uniform Sampling FPS : 13 ms per frame: 76ms radius = 0.3 σ = 0.040, k = 3	32 Samples Uniform Sampling FPS : 20 ms per frame: 50ms radius = 0.1 σ = 0.044, k = 3	0.0593	0.1164
64 Samples Stratified Sampling FPS : 13 ms per frame: 76ms radius = 0.3 σ = 0.064, k = 2	32 Samples Stratified Sampling FPS : 19 ms per frame: 52ms radius = 0.1 σ = 0.044, k = 3	0.0614	0.1216
64 Samples Cosine(θ) Sampling FPS : 13 ms per frame: 76ms radius = 0.3 σ = 0.010, k = 1	32 Samples Cosine(θ) Sampling FPS : 18 ms per frame: 55ms radius = 0.1 σ = 0.060, k = 2	0.1204	0.1205

FIGURE 4.21: Comparison between a set of resulted images from rendering living room with different radius and tuning parameters.

In this resulted comparisons we can see clearly the efficiency of the tuning parameters along the hemisphere radius in generating much more acceptable results even when using a 32 sample kernel size which makes performance more higher than using 64 kernel size along with gaussian blur which leads us to consider the tuning parameters along hemisphere radius and the sampling techniques a great way to produce and control the realism of the resulted outputs which also depends on the screen complexity when in comes to the fps calculation.

4.5.6 Filtering Techniques

In order to get rid of the noise that is within AlchemySSAO factor we gonna add a blur effect to the resulted texture or image in more accurate way and compare between the blurred AlchemySSAO factor from the different filtering techniques.

4.5.6.1 Bilateral Filter

With and Without bilateral filter are illustrated in the Figure 4.22 below using the following parameters:

- $\Rightarrow$ $\sigma_s = 3$ which is the spatial parameter.
- $\Rightarrow$ $\sigma_r = 3$ which is the range weight.

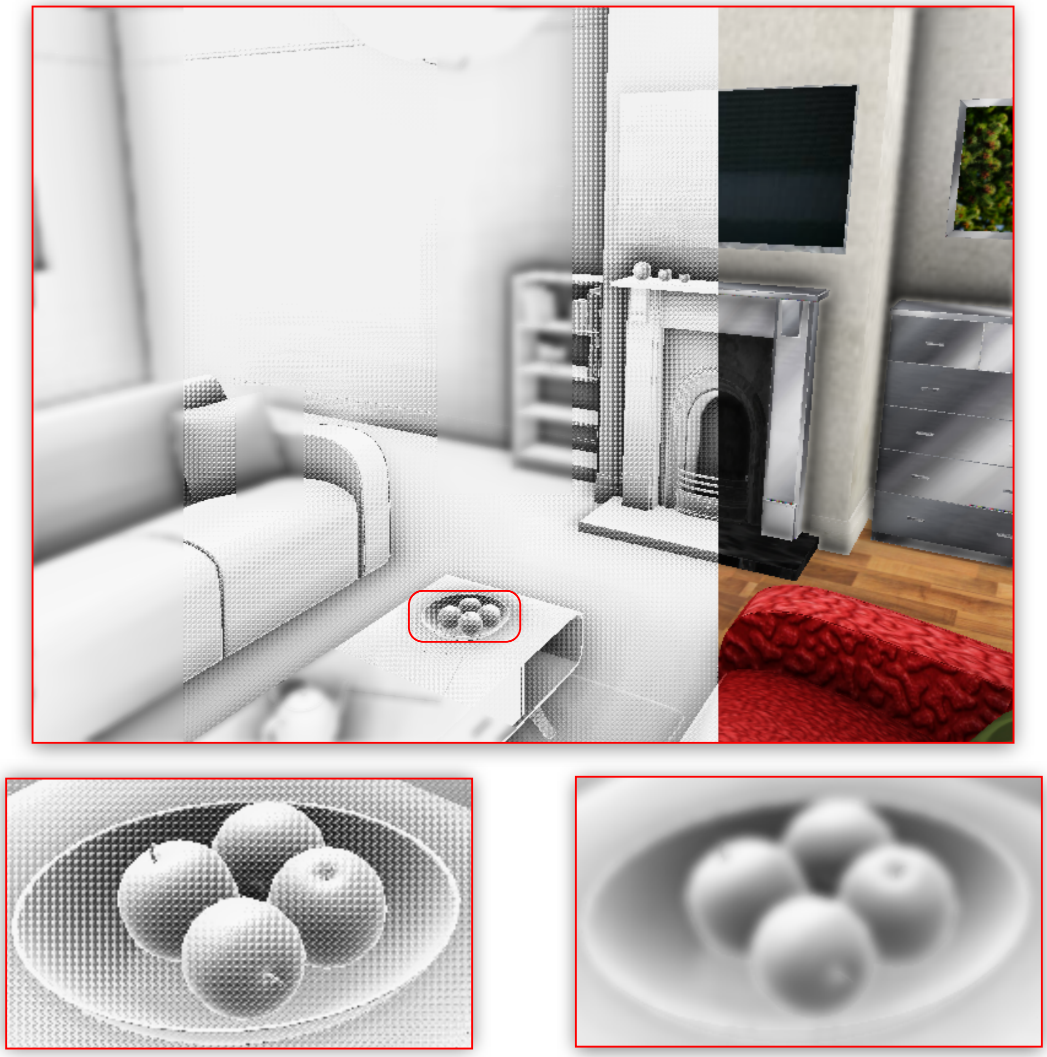


FIGURE 4.22: Blurred AlchemySSAO factor using the bilateral filter with **FPS: 9** and **ms per frame average: 111ms**, without **FPS: 13** and **ms per frame average: 76ms**

The resulted AlchemySSAO factor texture are clearly enhanced and the noisy spots are significantly vanished and the scene geometry shapes are preserved due the fact that the bilateral filter consist of edge preserving but when it comes to performance it is a time consuming technique.

4.5.6.2 Gaussian Filter

Another filter is implemented which is the gaussian filter where the blurring effect are splitted into two phases horizontal and vertical blurring as Figure 4.23 illustrate where the result are shown in Figure 4.24 with parameters:

⇒ Blur strength = 2.0.

⇒ Blur kernel size = 8.

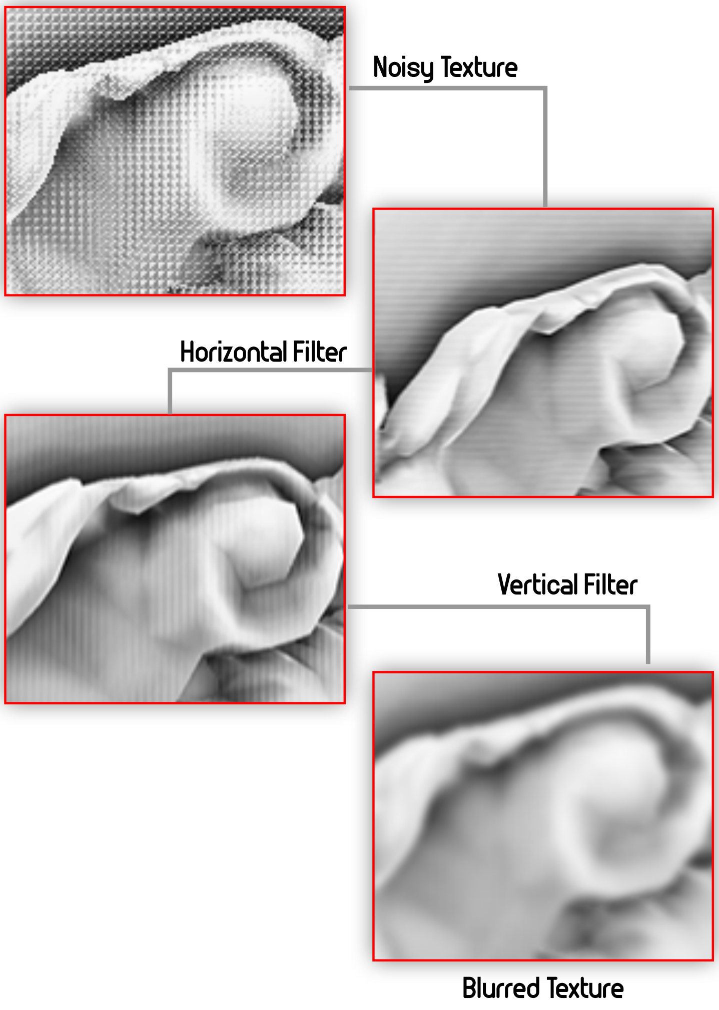


FIGURE 4.23: Gaussian filter blurring phases illustration.

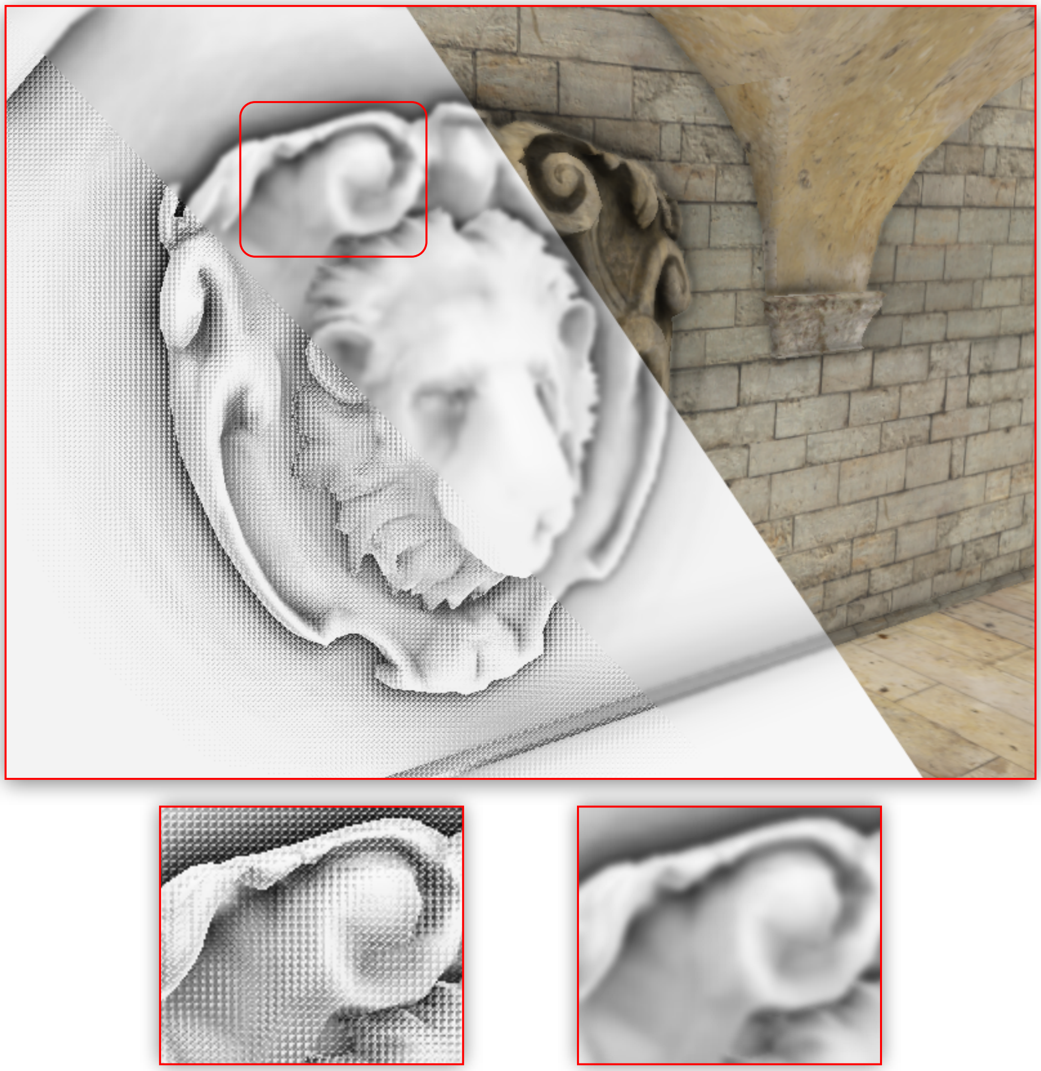


FIGURE 4.24: Blurred AlchemySSAO factor using the gaussian filter with **FPS: 17** and **ms per frame average: 58ms**, without **FPS: 23** and **ms per frame average: 43ms**

The gaussian filter gives a great results when it comes to noise reduction but it also must be well controlled in order to preserve scene geometry shapes due the fact that it only uses spatial parameter which makes edge preserving absent causing details loosing.

4.5.6.3 4x4 blur Filter

The last applied filter is the 4x4 blur filter which is illustrated in the Figure 4.25 below.

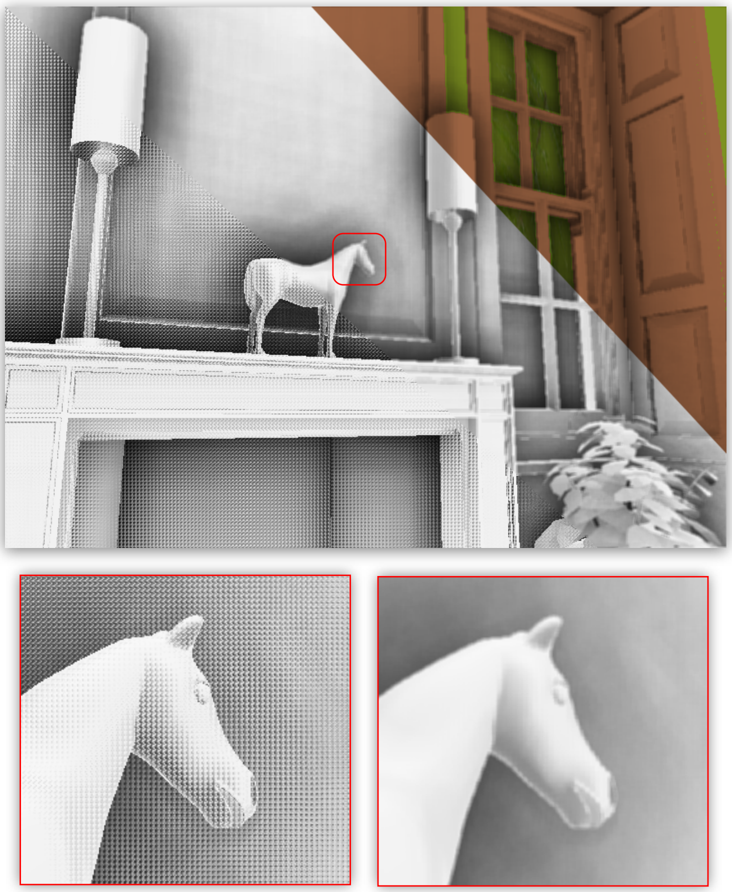


FIGURE 4.25: Blurred AlchemySSAO factor using the 4x4 blur filter with **FPS: 11** and **ms per frame average: 90ms**, without **FPS: 17** and **ms per frame average: 58ms**

A 4x4 blur considered to be an efficient way to provide a simple blurring that is used to increase performance as the Figure 4.25 above shows the difference in fps where it is considered to be small comparing to the gaussian and bilateral filters.

4.5.7 Filters comparison

The Figure 4.26 illustrate the comparison between the used filters and also the AlchemySSAO result without using filtering techniques.

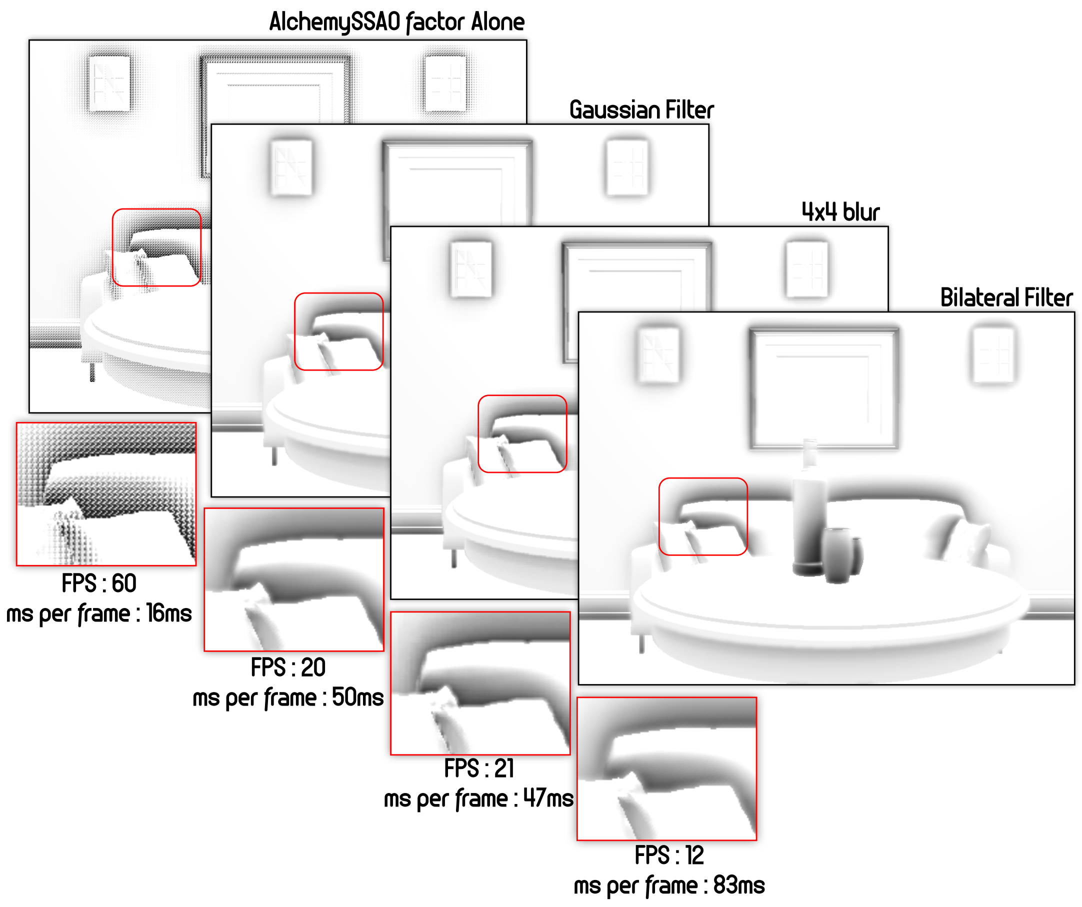


FIGURE 4.26: Comparison between the original resulted AlchemySSAO factor and the used filters

From the Figure 4.26 we can clearly see the decreasing of the fps the the increasing in the time taken for each frame where the bilateral filter considered to be as one of the low performance causes.

4.5.8 Scene Illumination

Scene illumination has been achieved using the Blinn-Phong illumination model where the Figure 4.27 shows the difference between an approximated ambient obscurance factor and a uniform ambient light value.

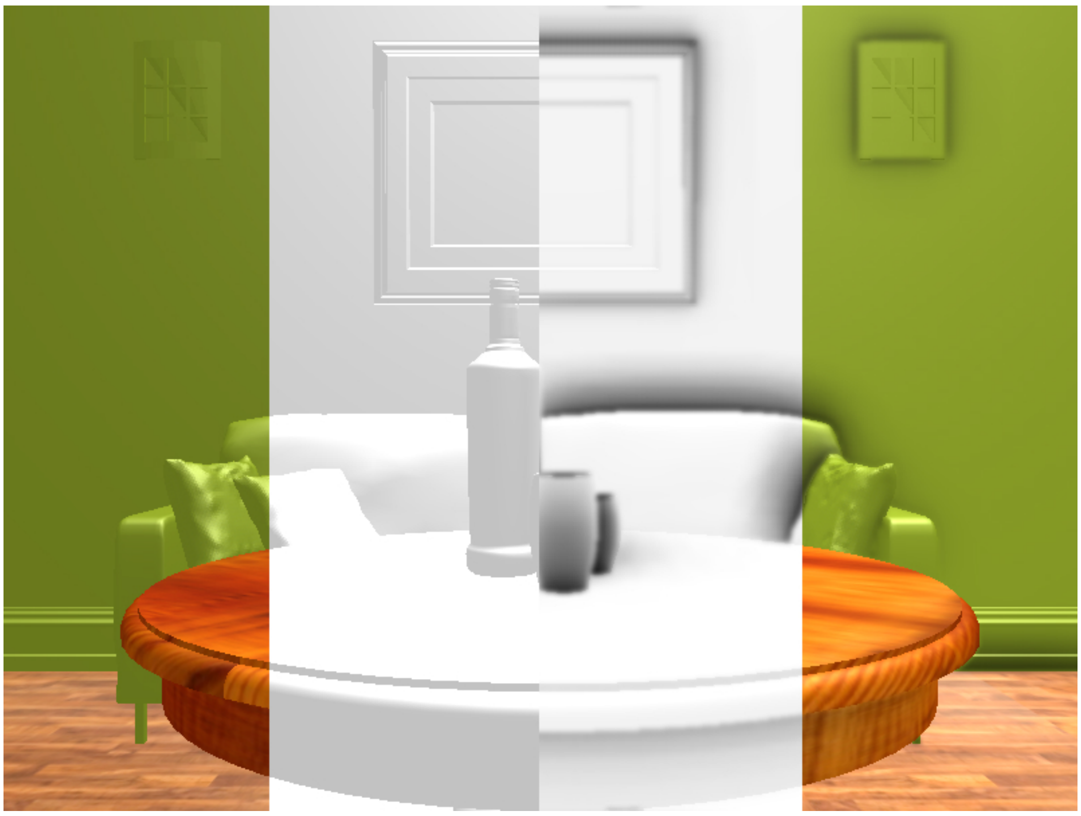
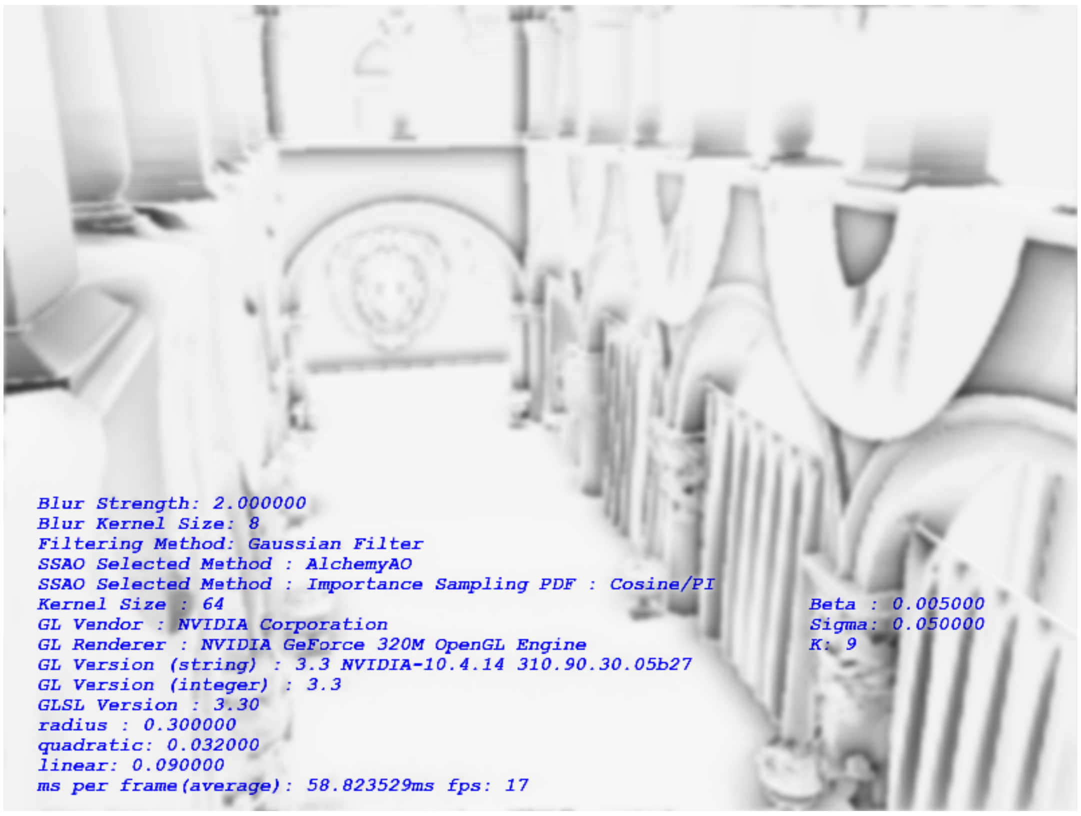


FIGURE 4.27: with AlchemySSAO factor **FPS: 20** and **ms per frame average: 50ms**, without **FPS: 60** and **ms per frame average: 16ms**

As the figure above illustrate the realism enhancement by using the AlchemySSAO to approximate the ambient obscurance factor where it effects significantly the blocked areas like behind the couch, between the cups, between the floor and the end of the wall, behind the picture frame where it is considered as one of the global illumination essences.

4.5.9 Text render

In order to dynamically check the tuning parameters for both AlchemySSAO and Filtering alongside with some of the materials and software Configurations a textual informations content projection are implemented as the Figure 4.28 shows.



```
Blur Strength: 2.000000
Blur Kernel Size: 8
Filtering Method: Gaussian Filter
SSAO Selected Method : AlchemyAO
SSAO Selected Method : Importance Sampling PDF : Cosine/PI
Kernel Size : 64                                                Beta : 0.005000
GL Vendor : NVIDIA Corporation                                  Sigma: 0.050000
GL Renderer : NVIDIA GeForce 320M OpenGL Engine                 K: 9
GL Version (string) : 3.3 NVIDIA-10.4.14 310.90.30.05b27
GL Version (integer) : 3.3
GLSL Version : 3.30
radius : 0.300000
quadratic: 0.032000
linear: 0.090000
ms per frame(average): 58.823529ms fps: 17
```

FIGURE 4.28: Tuning parameters alongside the materials and softwares configurations projected into the application frame.

4.5.10 Results Discussion

After doing a comparison between the sampling techniques using different scenes in order to observe the difference between the reference image and the different images resulted from sampling techniques by using fixed parameters and per-scene different parameters along the root mean squared error where each of them is considered as a metric for difference observation and also using the frame per second (FPS) and the noise degree in the resulted AlchemySSAO factor along the ms per frame average, we conclude that both the bilateral and randomness sampling technique which are used in the original authors in AlchemySSAO can be replaced either with a uniform or stratified distribution or even an importance sampling technique while the blurring can be enhanced when it comes to performance by using either a gaussian blur or a 4x4 simple blur and of course the controllable tuning, filtering parameters followed by the hemisphere radius makes instant realism enhancing possible, all of these components are combined together in order to achieve an acceptable ambient obscurance approximation using the screen space.

4.5.11 Conclusion

In this chapter, we did an implementation of the original AlchemySSAO technique with its both original used techniques random sampling and bilateral filtering that causes low performance when it comes to the bilateral filtering and noise producing along important samples ignorance, which is caused by the random sampling. Due to these problems, we tried to achieve an acceptable ambient obscurance factor approximation and good performance by using other different techniques rather than the original ones for filtering and sampling, where we used a gaussian and 4x4 blur as filtering techniques which provide a better performance. For sampling techniques we did a comparative study between three different distribution techniques along with a chosen importance sampling pdf that were compared either with fixed AlchemySSAO tuning parameters applied to five different scenes that allowed us to evaluate the amount of difference there is with a reference image, or with controllable parameters applied to three different scenes along with different samples kernel size to enhance performance and reduce the noise effect as much as we could, where all of these implementations were possible using the power of gpus when it comes to shaders programming let us include real-time factor in approximating one of the global illumination resulted effects which is the ambient obscurance.

General Conclusion

In this project, we have presented the foundation of the global illumination along with its different approximation methods that have different essentials. One of these methods is the ambient obscurance which is an extended version of the ambient occlusion, where it depends on scene geometry in order to determine the amount of obstruction there is between one point, or a region of a 3D scene and a light source whereby all geometry is taken into account which is not practical in real-time application.

Alchemy Screen Space Ambient Obscurance from the other hand is one of the solutions that is used to approximate the ambient obscurance in real-time, where it depends on the screen space more accurately pixel depth rather than the geometry itself which means that it uses the depth buffer as an approximation of the visible geometry which allows faster ambient obscurance approximations that can be used in real-time. But due to the used sampling and filtering techniques it is more exposed to low performance, and noise not to mention that it usually misses an important scenes contributions when it comes to sampling technique.

To solve this kind of problems, we implemented the alchemy screen space ambient obscurance using the original authors used sampling and filtering techniques which are the random sampling and bilateral filter respectively, and another set of chosen techniques which are the uniform, stratified sampling along the importance sampling with a $\cos(\theta)$ as a probability density function and a gaussian, 4x4 filters where the different results have been compared together using five different scenes with fixed tuning parameters, and three other scenes with controllable parameters in order to come up with a better results that has a better samples expectations, higher performance and lower noise. We also did a comparison with the original crytek screen space ambient occlusion that leads us to efficient results as we expected. All of this is accomplished by using the power of GPUs as the graphics essence.

For the future works, we suggest ideas that can enhance the outputs in much more better way like:

- ⇒ Using textures rather than vectors in samples transferring between the CPU and the GPU to accelerate the acces.
- ⇒ Using Mutiple Importance Sampling rather that different used techniques for better sampling.

References

[1] Zerari Abd El Moumene. “Volume D’ombre en Rendu Temps Réel”. MA thesis. Biskra: Mohamed kheidher University, Sept. 2011.

[2] Kelly Dempski and Emmanuel Viale. *Advanced Lighting and Materials with Shaders*. 2320 Los Rious Boulevard Plano, Texas 75074: Wordware, 2005. ISBN: 1-55622-292-0.

[3] Dave Shreiner et al. *OpenGL Programming Guide*. Pearson Education, One Lake Street Upper Saddle River, New Jersy 07458: Edwards Brothers Malloy, 2013. ISBN: 978-0-321-77303-6.

[4] Edward Angel and Dave Shreiner. *Interactive Computer Graphics*. 7th ed. Pearson Education, One Lake Street Upper Saddle River, New Jersy 07458: Addison-Wesley, 2015. ISBN: 978-0-13-357484-5.

[5] Bui Tuong Phong. “Illumination for Computer Generated Pictures”. In: *Commun. ACM* 18.6 (June 1975), pp. 311–317. ISSN: 0001-0782. DOI: 10.1145/360825.360839. URL: http://doi.acm.org/10.1145/360825.360839.

[6] James F. Blinn. “Models of Light Reflection for Computer Synthesized Pictures”. In: *SIGGRAPH Comput. Graph.* 11.2 (July 1977), pp. 192–198. ISSN: 0097-8930. DOI: 10.1145/965141.563893. URL: http://doi.acm.org/10.1145/965141.563893.

[7] Elhadad Feriel. “Photon Mapping Acceleration for global rendering”. MA thesis. Biskra: Mohamed kheidher University, Sept. 2013.

[8] Ritschel Tobias et al. “The State of the Art in Interactive Global Illumination”. In: *Computer Graphics Forum* 31.1 (Feb. 2012), pp. 160–188. DOI: 10.1111/j.1467-8659.2012.02093.x. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2012.02093.x. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2012.02093.x.

[9] Townsend Tamsyn. *Ray Tracing*. University Lecture. 2017.

[10] Ian Ashdown. *Radiosity: A Programmer’s Perspective*. New York, NY, USA: John Wiley & Sons, Inc., 1994. ISBN: 0471304433.

[11] Duck Bong Kim et al. “High-dynamic-range camera-based bidirectional reflectance distribution function measurement system for isotropic materials”. In: 48 (Sept. 2009).

[12] Matt Pharr and Greg Humphreys. *Physically Based Rendering, Second Edition: From Theory To Implementation*. 2nd. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2010. ISBN: 0123750792, 9780123750792.

[13] James T. Kajiya. "The Rendering Equation". In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 143–150. ISSN: 0097-8930. DOI: 10.1145/15886.15902. URL: http://doi.acm.org/10.1145/15886.15902.

[14] Rita Zrour. "Global Illumination method parallelism using Voxles". PhD thesis. Auvergne University, 2007.

[15] Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. "Combining Hierarchical Radiosity and Discontinuity Meshing". In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. Anaheim, CA: ACM, 1993, pp. 199–208. ISBN: 0-89791-601-8. DOI: 10.1145/166117.166143. URL: http://doi.acm.org/10.1145/166117.166143.

[16] REDWAY3D. *Monte Carlo Sampling*. Accessed: 2018-05-26. 2017. URL: http://www.redway3d.com/downloads/public/documentation/bk_re_monte_carlo_sampling.html.

[17] Peter Shirley and Steve Marschner. *Fundamentals of Computer Graphics*. 4rd. Natick, MA, USA: A. K. Peters, Ltd., 2016. ISBN: 1568814690, 9781568814698.

[18] Eric Veach. "Robust Monte Carlo Methods for Light Transport Simulation". PhD thesis. Standford University, 1997.

[19] Konstantinos Vardis. "Efficient Illumination Algorithms for Global Illumination In Interactive and Real-Time Rendering". PhD thesis. Athens University, 1997.

[20] Alexander Keller. "Instant Radiosity". In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 49–56. ISBN: 0-89791-896-7. DOI: 10.1145/258734.258769. URL: https://doi.org/10.1145/258734.258769.

[21] Samuli Laine et al. "Incremental Instant Radiosity for Real-Time Indirect Illumination". In: *Rendering Techniques*. 2007.

[22] Carsten Dachsbacher and Marc Stamminger. "Reflective Shadow Maps". In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D '05. Washington, District of Columbia: ACM, 2005, pp. 203–231. ISBN: 1-59593-013-2. DOI: 10.1145/1053427.1053460. URL: http://doi.acm.org/10.1145/1053427.1053460.

[23] T. Ritschel et al. "Imperfect Shadow Maps for Efficient Computation of Indirect Illumination". In: *ACM Trans. Graph.* 27.5 (Dec. 2008), 129:1–129:8. ISSN: 0730-0301. DOI: 10.1145/1409060.1409082. URL: http://doi.acm.org/10.1145/1409060.1409082.

[24] Oliver Mattausch. "Visibility Algorithms for Real-Time Rendering in General 3D Environements". PhD thesis. Vienna University of Technology, 2010.

[25] David Wolff. *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing, 2011. ISBN: 1849514763, 9781849514767.

[26] Eisemann. Elmar et al. *Real-Time Shadows*. A K Peters/CRC Press, 2011. ISBN: 978-1-56881-438-4.

[27] Hayden Landis. “Production-ready global illumination”. In: *Siggraph course notes* 16.2002 (2002), p. 11.

[28] Wolfgang Engel. “ShaderX7”. In: *Charles River Media* (2009).

[29] Frederik Peter Aalund. “A Comparative Study of Screen Space Ambient Occlusion Methods”. MA thesis. Denmark: Technical University of Denmark Informatics and Mathematical Modelling, May 2013.

[30] Sergey Zhukov, Andrei Iones, and Grigorij Kronin. “An ambient light illumination model”. In: *Rendering Techniques’ 98*. Springer, 1998, pp. 45–55.

[31] Marc Sunet. “Ambient Occlusion on Mobile: An Empirical Comparison”. MA thesis. Spain: POLYTECHNIC UNIVERSITY OF CATALONIA, 2016.

[32] Janne Kontkanen and Samuli Laine. “Ambient Occlusion Fields”. In: *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*. I3D ’05. Washington, District of Columbia: ACM, 2005, pp. 41–48. ISBN: 1-59593-013-2. DOI: 10.1145/1053427.1053434. URL: http://doi.acm.org/10.1145/1053427.1053434.

[33] Mattias Malmer et al. “Fast precomputed ambient occlusion for proximity shadows”. In: *Journal of graphics tools* 12.2 (2007), pp. 59–71.

[34] Matt Pharr and Randima Fernando. *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005.

[35] Mirko Sattler et al. “Hardware-accelerated ambient occlusion computation”. In: *Vision, Modeling, and Visualization 2004*. Ed. by B. Girod, M. Magnor, and H.-P. Seidel. Akademische Verlagsgesellschaft Aka GmbH, Berlin, Nov. 2004, pp. 331–338. ISBN: 3-89838-058-0.

[36] Martin Mittring. “Finding next gen: Cryengine 2”. In: *ACM SIGGRAPH 2007 courses*. ACM. 2007, pp. 97–121.

[37] Tobias Ritschel, Thorsten Grosch, and Hans-Peter Seidel. “Approximating dynamic global illumination in image space”. In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM. 2009, pp. 75–82.

[38] Quagliozzi Eric. *Z buffering*. Accessed: 2018-05-31. 2017. URL: http://www.pda-fx.net/pagedoc.php?id=4&lg=EN.

[39] Dominic Filion and Rob McNaughton. “Effects & techniques”. In: *ACM SIGGRAPH 2008 Games*. ACM. 2008, pp. 133–164.

[40] Louis Bavoil, Miguel Sainz, and Rouslan Dimitrov. “Image-space horizon-based ambient occlusion”. In: *ACM SIGGRAPH 2008 talks*. ACM. 2008, p. 22.

[41] Bradford James Loos and Peter-Pike Sloan. “Volumetric obscurance”. In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM. 2010, pp. 151–156.

[42] Morgan McGuire et al. “The alchemy screen-space ambient obscurance algorithm”. In: *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. ACM. 2011, pp. 25–32.

[43] Nvidia. *Anisotropic Filtering*. Accessed: 2018-06-01. 2012. URL: https://www.geforce.com/whats-new/guides/aa-af-guide#1.

[44] Quora. *What consumes more CPU power in regard to games - anisotropic filtering or anti-aliasing*. Accessed: 2018-06-01. 2017. URL: https://www.quora.com/What-consumes-more-CPU-power-in-regards-to-games-anisotropic-filtering-or-anti-aliasing.

[45] Guang Deng and LW Cahill. "An adaptive Gaussian filter for noise reduction and edge detection". In: *Nuclear Science Symposium and Medical Imaging Conference, 1993., 1993 IEEE Conference Record.* IEEE. 1993, pp. 1615–1619.

[46] Sylvain Paris et al. "Bilateral filtering: Theory and applications". In: *Foundations and Trends® in Computer Graphics and Vision* 4.1 (2009), pp. 1–73.

[47] Graham Sellers, Richard S. Wright, and Nicholas Haemel. *OpenGL Super-Bible: Comprehensive Tutorial and Reference*. 6th. Addison-Wesley Professional, 2013. ISBN: 0321902947, 9780321902948.

[48] Prashanta Kumar Das and Ganesh Chandra Deka. "History and evolution of gpu architecture". In: *Emerging Research Surrounding Power Consumption and Performance Issues in Utility Computing*. IGI Global, 2016, pp. 109–135.

[49] Tomas Akenine-Moller, Eric Haines, and Naty Hoffman. *Real-time rendering*. AK Peters/CRC Press, 2008.

[50] Joey de Vries. *Learn OpenGL*. Joey de Vries, 2017.

[51] Morgan McGuire. *Computer Graphics Archive*. Accessed: 2018-06-05. 2017. URL: https://casual-effects.com/data.