HAMK
Hämeen ammatti-
korkeakoulu

REACHING PHOTOREALISM WITHIN THE LIMITATIONS OF REAL-TIME COMPUTER GENERATED GRAPHICS

Reaching towards photorealism on Unreal Engine 5

Bachelor's thesis
Information and Communication Technology
Autumn, 2024
Ville-Matti Pursiainen

HAMK
Hämeen ammatti-
korkeakoulu

Information and Communication Technology Abstract
Author Ville-Matti Pursiainen Year 2024
Subject Reaching Photorealism Within the Limitations of Real-Time Computer Generated Graphics
Supervisor Mika Virolainen

The purpose of this thesis is to present a workflow and methods of rendering photorealistic real time visualizations and video games. It explores methods that can add layers of realism to real-time computer-generated graphics through real world phenomena, how light behaves and how humans perceive things.

Achieving photorealism requires simulating, and often approximating, these real-world aspects according to the behaviour of light and surfaces. This involves understanding how light interacts with surfaces, how shadows form, and how materials appear under various lighting conditions. Human perception is remarkably adept at detecting inconsistencies in visual scenes; even minor deviations from reality can render a digital image unconvincing.

The theoretical section goes through elements that can help to create more realism in digital visualization. It discusses topics of how lighting is imitated digitally in video game engines and what are their features and limitations in regards of lighting and how it behaves on different mediums artificially. It compares Unreal engine 5's ability to execute these techniques and how it approaches them.

The methodological approach of this thesis uses the elements explored in the theoretical section to create the workflow for reaching towards realism for the visualization in unreal engine 5 and going into depth of its graphical capabilities to make photorealistic visualization of a modern luxury house. Blender is used as a tool alongside to achieve the ultimate result.

The result is a fairly photorealistic visualization of a modern luxury house and acts as a great example of how Unreal Engine 5 can be used to make real-time visualizations.

Keywords Photorealism, Unreal Engine 5, Visualization, Real-time graphics

Pages 25 pages and appendices 8 pages

Contents

List of figures

List of appendices

1 Introduction

The subject of this thesis is to study what methods can be used to help reach towards photorealism in computer generated real-time graphics. Interest in the subject grew from a hobby of making artistic visualizations with blender and some experimental games with different game engines including Unity, Unigine, Godot and Unreal Engine 3, 4 and 5. They all tangled in-between a blend of unique art-style and realism. This thesis aims to present a workflow and methods to help reach towards photorealism on video game engines.

Video game graphics are an ever-evolving subject. Realism and art-style are always subjects that are strongly present in video game industry and among gamers. Recently Unreal Engine has also been utilized in making movies and TV series either with led walls presenting scenery backdrop or as general embedded computer generated graphics or CGI for short. For example the TV series The Mandalorian on Disney+ and Fallout on Amazon Prime were filmed this way (Farris, 2020; Epic Games, 2024). The subject of the backdrops and CGI being real time emerges in these writings from Farris and Epic Games making it easier and cheaper than ever before to produce quality environments for the films. Therefore photorealism is currently an important aspect to reach in video game engines. Filmmakers academy (2021) ponders that the entertainment industry's use of real time video game engine's CGI together with real life actors and set pieces is gaining traction and that it is the future of film making.

This thesis aims to answer what key layers are needed for photorealism with real time video game engine graphic on static scenes. Aspects of dynamics and movement are also key elements of realism but are left out of this thesis as subjects.

2 Layers of realism

Before creating advanced computer generated imagery (CGI) that strives for realism, it is crucial to understand the visual characteristics of the real world. This includes light's behaviour and how humans perceive it. The natural world encompasses intricate elements such as illumination, shading, shapes, and textures that must be accurately replicated in digital imagery.

Achieving photorealism requires simulating, and often approximating, these real-world aspects according to the behaviour of light and surfaces. This involves understanding how light interacts with surfaces, how shadows form, and how materials appear under various lighting conditions. Art-style is often way more important aspect to grasp into in videogames than realism because human perception is remarkably adept at detecting inconsistencies in visual scenes; even minor deviations from reality can render a digital image unconvincing. However, it does not mean that realism and art-style cannot work hand in hand.

The theory section will explore these core elements and discuss why understanding light, shadow, and material interaction is essential for photorealism and how Unreal Engine 5 implements to achieve such details in real time. These are the core aspects of what creates realism in digital graphics.

2.1 Shape and geometry

Geometry together with lighting are two of the most important parts of realism. Shape, its topology and geometry are crucial cues for object recognition and to an extent the whole world around us. Shape refers to the overall appearance of the object's topology's relationship between points in space and their arranged connections to each other calculated by precise mathematical definitions and formulas of geometry to describe these properties. (Morgenstern et al., 2021, s. 2)

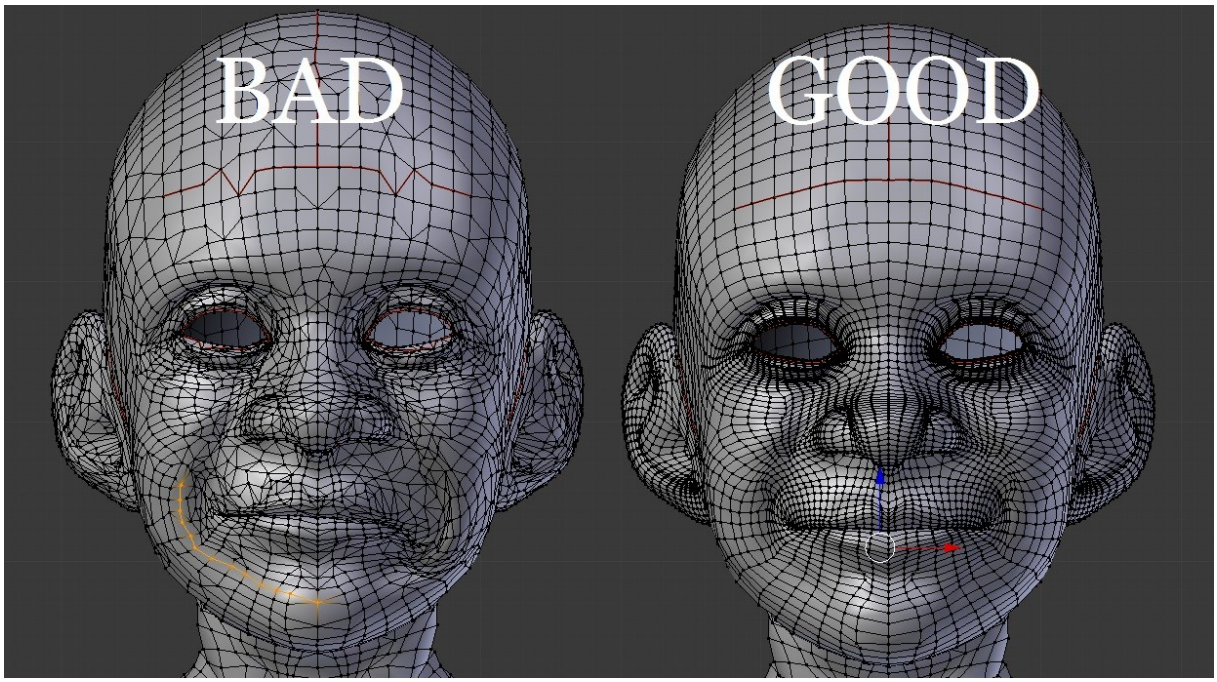
In video games, geometry is often the foundation even in 2d-games. Either a game about jumping from a pillar to pillar in a 2d-video game or hiding from enemy around the corner of a shooter game are based largely on geometry. Real life objects often have very complex geometry that is difficult to represent in just a few polygons (Math worksheets center, n.d.). With a good topology and a few lighting gimmicks we can have performant geometry that looks good and complex. One of these gimmicks are normal map textures that are explored further later in this thesis.

2.1.1 3D modeling

Planning of the topology and geometry of the 3D models is very important during modeling and even before it. Especially when modeling an asset to a game, it is good to take all

possible optimizations into account. An approach to achieve good topology can vary between polygonal modeling and curve modeling for example and there are various methods of using them to get desirable edge flows. A desirable edge flow means there is a substantial number of adjacent square polygons. Triangle and n-gons break the topology edge flow. Figure 1 demonstrates desirable edge flow and topology compared to inadequate one. Desirable edge flow eases the selection of faces on model allowing long chain of adjacent triangle faces to be selected and also gives cleaner bending of the polygons when animated. (Martin, 2021)

Figure 1. Example of inadequate vs desirable topology of 3D model of a face. (Jacks blog, 2018)



2.1.2 Low-poly

Polycount is very important in video-game asset modeling. The smaller the polycount the better the performance. The key to making photorealistic or detailed asset while maintaining low polycount is to first model low poly object with good topology then continue to model detailed high poly duplicate of the same object and bake the details from the high poly to low poly object's normal maps. (Arm Limited, 2020)

2.1.3 Sculpting

Sculpting is a good choice when a lot of small geometric detail is needed in a 3D asset. It is a digital technique to sculpt the model as it were clay. Usually, software that offer digital sculpting come with at least inflating, deflating, pinching and smoothing tools. Figure 2 demonstrates an example of the use of virtual sculpting. When a desired shape has been modelled with low polycount, the object can be cloned and the cloned object can be added with subdivision modifiers. This increases the polygon count of that object which makes possible to sculpt detailed geometry to it. Some softwares offer dynamic topology, which increase the polycount per pixel on the site of current sculpting brush. It is also possible to first sculpt high-poly model and then decimate the geometry to lower polycount but that way the good topology frow is sacrificed. If topology is essential for the asset it is advised to first model it in the conventional polygon modeling way and then sculpt on the cloned model. Images can also be used as a brush of the sculpting where grayscale controls the strength of deflation / inflation. (Heginbotham, n.d.)

Figure 2. Dragon’s head being sculpted using blender’s sculpting tools.

2.1.4 Photogrammetry

Photogrammetry is a method of acquiring accurate data about properties of physical surfaces and objects in real world with for example camera lidar or laser scanner. According to Schenk (2005), photogrammetry data consist of geometric information, physical information, sematic information and temporal information. The geometric information provides spatial positioning and data for the shape of photogrammed objects and their surfaces. The physical information layer addresses electromagnetic radiation properties for optimized image capture and surface properties such as reflectance.

2.1.5 Unreal engine 5's solution to near unlimited geometric detail

Epic Games (n.d.-a) state that Nanite virtualized geometry is the Unreal Engine 5's solution to poor performance on highly detailed meshes with lots of polygons. It achieves high performance by breaking the mesh down to clusters and during rendering the clusters are broken down and displayed as sub-clusters on a pixel scale.

2.2 Texturing detail

Detailed texturing can add a lot to realism when done within good limits. Careful detail placement can fake details that geometry cannot. These details can for example be additional faked geometry using normal maps, microscopic surface inconsistency using roughness maps and intricate colorful diversification using albedo maps.

2.2.1 Normal map

The name normal mapping comes from surface normals that are used to calculate and fake additional geometry on an object by tricking the rendering engine to bounce light according to the object's faked additional geometry surface normal. The additional geometry is presented as data from normal map texture, even though the real geometry would be flat. Figure 3 and 4 demonstrate how normal maps can help fake geometry in an object. The left speaker in Figure 3 has normal map enabled. It has been baked from the right speaker which has real geometry on its front face. In Figure 4 the normal map is disabled on the same speaker. In both figures, the right speaker depicts the real geometry.

Figure 3. Normal map on.

Figure 4. Normal map off.

2.2.2 Roughness

Roughness of a 3D model can be defined using a grayscale image. Usually the whiter the image, the more rough the surface is and on the contrary the darker the image the smoother the surface is. Some rendering software have it the other way round and call it smoothness instead of roughness texture. Artists can therefore create quite convincing detailed surfaces to materials with roughness map textures.

2.2.3 Stencil painting

Stencil texturing is one of the good ways to get realistic results in texturing a 3D model asset. It works by compositing one texture from many textures by painting the model through existing texture creating varied composition. (Blender Foundation, 2024)

2.3 Rendering and Material properties

2.3.1 Physically based rendering and its implementation in Unreal engine 5

The surface material of the 3D model is huge part of the realism. PBR or physically based rendering materials have become more common in game engines which have impacted the visual realism in games in a positive way. Pbr material is based on BSDF (Bidirectional Scattering Distribution Function) which describes how light scatters from hitting a surface. Microfacet theory, that the BSDF follows, suggests that all surfaces at microscopic scales are perfectly reflective little mirrors called microfacets which vary in alignment based on roughness of the surface. The more rough the surface is, the more of the microfacets are aligned on different directions from each other. The microfacet model also covers energy conservation of light hitting the surface. This means that light is split to diffuse and specular light once hitting the material's surface. The light rays that hit the surface and directly get reflected without getting absorbed in are called reflection rays and they form the specular light. The ones that got absorbed by the surface before getting reflected back are called refracted rays. They form the diffused light. (Dunn & Wood, n.d.; Vries, n.d.)

The history of shading models is long starting from pioneers like Henri Gourad and Bui Tuong Phong proposing a method to compute how reflections occur on surfaces based on their spatial relationships with light sources. Later, Jim Blinn built upon Ken Torrence's work on inter-reflections at the microscopic level to develop a reflection model that evolved into popular Cook-Torrence shading model (see Figure 5) by Rob Cook in collaboration with Ken Torrence himself. (Whitted, 2018)

According to Unreal engine 5's official documentation the shading model is largely the same as unreal engine 4's because it references unreal engine 4's Real shading in Unreal Engine 4 by Karis (2013). The Unreal engine 5 implements the physically based shading by using a Cook-Torrance microfacet specular shading model.

Figure 5. The Cook-Torrance microfacet specular shading model used in Unreal engine 5 (Karis, 2013, p. 3)

$$f(\mathbf{l}, \mathbf{v}) = \frac{D(\mathbf{h})\, F(\mathbf{v}, \mathbf{h})\, G(\mathbf{l}, \mathbf{v}, \mathbf{h})}{4\,(\mathbf{n} \cdot \mathbf{l})\,(\mathbf{n} \cdot \mathbf{v})}$$

2.3.2 Albedo

Albedo is the reflected radiation of surface hit by light. It defines the light intensity and color of the reflected light from the surface of PBR material. According to Weisstein (n.d.) it is calculated by dividing incident power of total scattered power of light radiation. In computer graphics it means how much of the diffused light the surface is reflecting. It is not to be confused with specular reflection. (Hukseflux, n.d.)

If some surface had an albedo value of 1 it would reflect 100% of the light shined to it. That would of course be unrealistic. Every surface on earth absorbs at least some light.

For example, if surface material albedo is 0,7 or 70%, 30% of light gets absorbed by the surface. It implies that the objects cannot be brighter than the light hitting them. In other words, albedo is the intensity in which different wavelengths of light is reflected of the surface. Typically, the base color value of PBR materials refer to albedo value.
(Geupel, 2022)

2.3.3 Specularity and its implementation in Unreal engine 5

Specularity is the angular distribution of the light reflected from material surface. It controls the angle of the surface the incident light reflects back. It is the amount of light that is reflected without the incident light penetrating the surface it is bouncing off. If a light particle penetrates the material's surface first before being reflected off after a few bounces inside the surface, the light is then called diffused reflection. Then again if it bounces straight back it is called specular reflection. Fresnel is also related to specularity where the lower the angle the camera is looking at the surface, the more specularity the surface has.
(Marsh, 2010; Nanda, 2023)

How Unreal engine 4 and 5 and most other software approach the implementation of this lighting phenomenon is by solving the microfacet specular shading model's normal distribution function using GGX/Throwridge-Reitz method. Schlick's approximation was used for solving the model's specular geometric attenuation in unreal engine 4.
(Karis, 2013, pp. 2–3)

A value between 0 and 1 is used to control specularity in material. Textures can also control the specular by their value of pixels just like roughness. Pitch black pixel is value of zero and pure white is 1 meaning 100% specularity.

2.4 Lighting

According to Freudenrich (2013, pp. 1–2) portraying light as rays allows for easy explanation of the phenomena that are reflection, refraction, and scattering. They are the three light behaviours that help us understand how light interacts in different situations. Reflection occurs when a light ray hits a smooth surface, a mirror for example, and bounces off from it. The bounced back ray has the same angle related to surface as the incident ray before hitting the surface. Physicists call this the law of reflection and express it as "the angle of incidence equals the angle of reflection". (Freudenrich, 2013, p. 2)

Scattering occurs when light rays hit rough surfaces, such as paper or rough rocks. When light hits these surfaces, it is scattered in all directions due to little microscopic angle changes all over the surface. These imperfections make the surface appear rough to the human eye because the bounced back picture is decoherent compared to incident ray.

Refraction occurs when light rays pass through a medium with different densities. The speed of light changes as it passes through different mediums, such as air or glass. This change in speed causes the light to bend and refract, which makes a straw in glass full of water appear to bend in Figure 6.

Figure 6. Straw appearing to bend demonstrating refraction.

Lighting and the implementation of it are one of the corner stones of realism in digital visualization. When lighting is implemented and executed thoughtfully and deliberately environment with even the simplest of details can be atmospheric and visually appealing.

It is however extremely heavy to calculate the simulation of all the real-life behaviours of light as real time computer graphics.

When hitting a surface, light gets either transmitted, reflected, absorbed, refracted, polarized, diffracted or scattered based on material of the surface. To implement these behaviours of light as computer graphics yet alone as a real time representation, a few gimmicks need to be deployed. The representation of light can be digitally implemented by the use of rasterization or ray tracing.

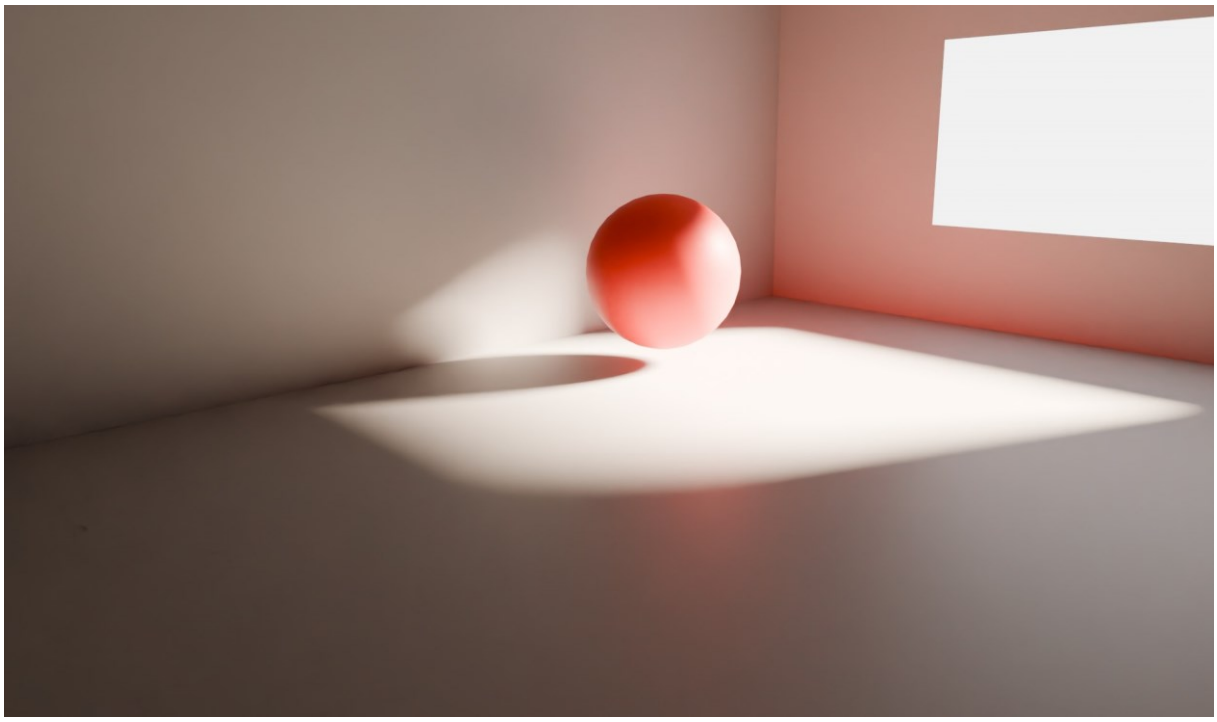
2.4.1 Global illumination

Global illumination or indirect lighting as the name suggests illuminates direct light indirectly whet hitting a surface. There are many ways that indirect lighting can be implemented in game-engines. For example, Lambru et al., (2021, pp. 4–5) lists that there are reflective shadow map, discrete ordinate method, voxel based method and screen space techniques. Usually indirect lighting needs to be pre-baked, which has its limitations. Prebaked lightmaps cannot influence their indirect lighting on dynamic objects. There are some gimmicks to that however. For example Unreal engine 4 offers dynamic real time global illumination through a method called light propagation volumes (Epic Games, n.d.-c), Godot offers signed distance fields (Linietsky & Manzur, n.d.) and unity light probes (Unity technologies, n.d.). They are all faking indirect lighting on rasterized graphics and work faster that ray traced solutions. Pre-baking lighting also has its benefits. It has second to none performance impact on runtime. A more accurate but not so performance friendly solution is to use ray-tracing algorithms. Ray tracing is really demanding to calculate because it is based on simulating how a real electromagnetic radiation would act, in this case a light ray. It is based on Kajiya equation. (Kajiya, 1986, p. 143)

In 1986, James Kajiya introduced a technique which included a few other known rendering algorithms. It has come to be known as the rendering equation. It is based on joules as the light intensity. The rendering equation became the base for ray- and path tracing. (Kajiya, 1986, pp. 143 – 144)

Path-tracing and ray tracing are essentially the same. Instead of a one ray per pixel method of ray tracing that always hits a lightsource, path-tracing calculates multiple rays per pixel with multiple samples per ray per pixel. Each ray bounces in random directions and will not always hit a lightsource. (Evanson, 2023). Figure 7 demonstrates indirect global illumination rendered with path traced rays.

Figure 7. Demonstration of global illumination.



2.4.2 Reflection

When light hits surface, it is bounced and depending on the roughness of the surface it appears either diffused or mirror like surface. Rasterized lighting cannot present this as it is because it cannot bounce the light from surface, it has to be faked. One of these fake hacks is to represent reflections as cubemaps that the surface appears to be reflecting. These cubemaps often have to be previously calculated from the given environment the surface is in. They are static as in they cannot be changed in runtime. There are however real-time reflection probe cubemaps that are achieved by rendering 360 camera from around the environment and displaying the rendered reflected image on surface of the material with material properties like roughness, specularity, normal and albedo distortion. (Unity Technologies, 2024)

A more physically accurate reflection in games is achieved through ray- or path tracing just like how global illumination can be done. Tracing rays from camera to surface and saving the values of the light bounce.

2.4.3 The Unreal Engine 5's solution to real-time global illumination in video games

The Unreal Engine 5 uses a system called Lumen for its global illumination and reflections. It is their own in-house developed solution that offers better performance than traditional path- or ray tracing and more realistic light calculation than rasterization-based lighting solutions. It is a form of ray tracing but differs from the traditional approach. Instead of shooting rays from the camera to a whole scene, Lumen uses something called surface caches to efficiently measure lighting at ray hit points in the scene with material properties captured from multiple angles. These automatic parameterizations of surface cache are called Cards. Cards capture material properties of each mesh from multiple angles and are far more efficient to ray trace against instead of original mesh geometry. For lumen to generate the surface cache and cards from it, it first needs mesh signed distance field data. It needs to be enabled in every mesh and is by default generated on mesh import. Lumen offers the feature to first ray-trace in screen space and when ray goes offscreen Lumen reverts to surface cache, however if the hardware supports hardware ray-tracing ray hit against triangle geometry from the original mesh itself is used. (Wright et al., 2022; Epic Games, n.d.-b)

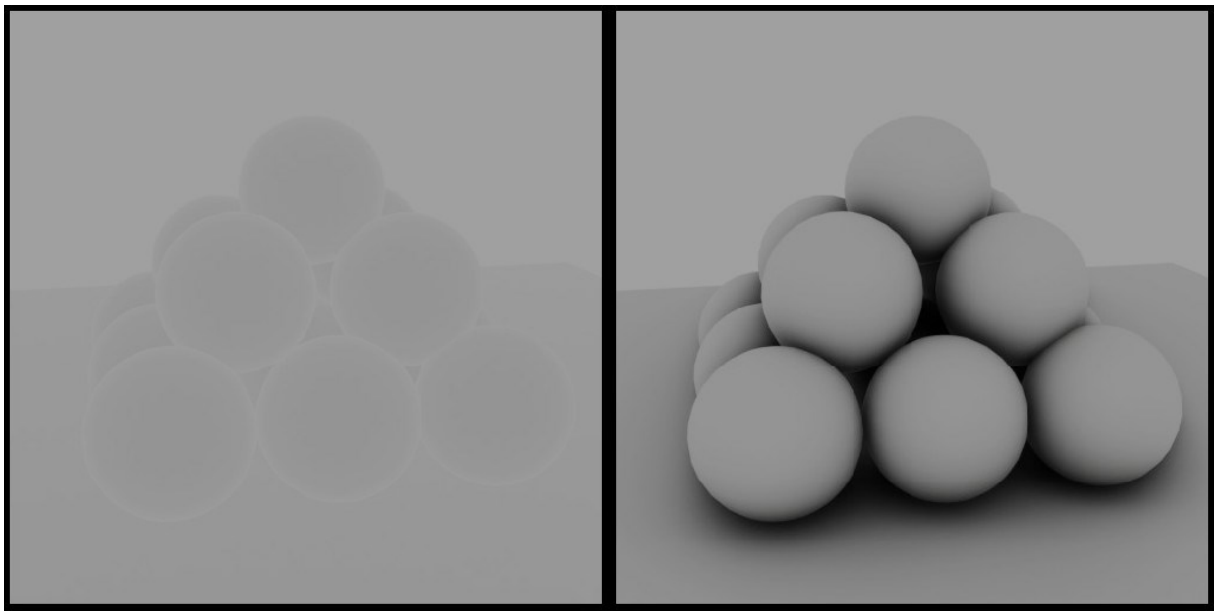
2.4.4 Shadows

The method of representing shadows in real-time computer graphics have gone through many revisions in the past decades. Like indirect lighting, shadows also have to be faked by shadow mapping techniques if they are not rendered with ray tracing. The fundamentals of how shadow mapping works is by saving objects z values in the scene for each pixel from the perspective of the light source. Then as rendered from the perspective of the camera before shading values each pixel is tested if it intersects shadow from light source space and is then shaded accordingly (Williams, n.d.). Two of the most used methods are simple shadow maps often referred to as ssm and cascaded shadow maps often referred to as csm.

2.4.5 Ambient Occlusion

According to Learn OpenGL (n.d.) Ambient occlusion is a self-shadowing approximation that reduces the lighting from indirect light sources in closely located geometry in greases and holes for example. It is usually calculated in screen-space by calculating occlusion factor on each observed points and saving their depth values based on surrounding geometry. The observed points with more depth than the averaged surrounding observed depth value contribute towards more occluded space in geometry. The more occluded the point in geometry is the more darker the indirect light is in that spot. Figure 8 shows ambient occlusion on and off.

Figure 8. A stack of spheres with ambient occlusion off (left) and on (right).



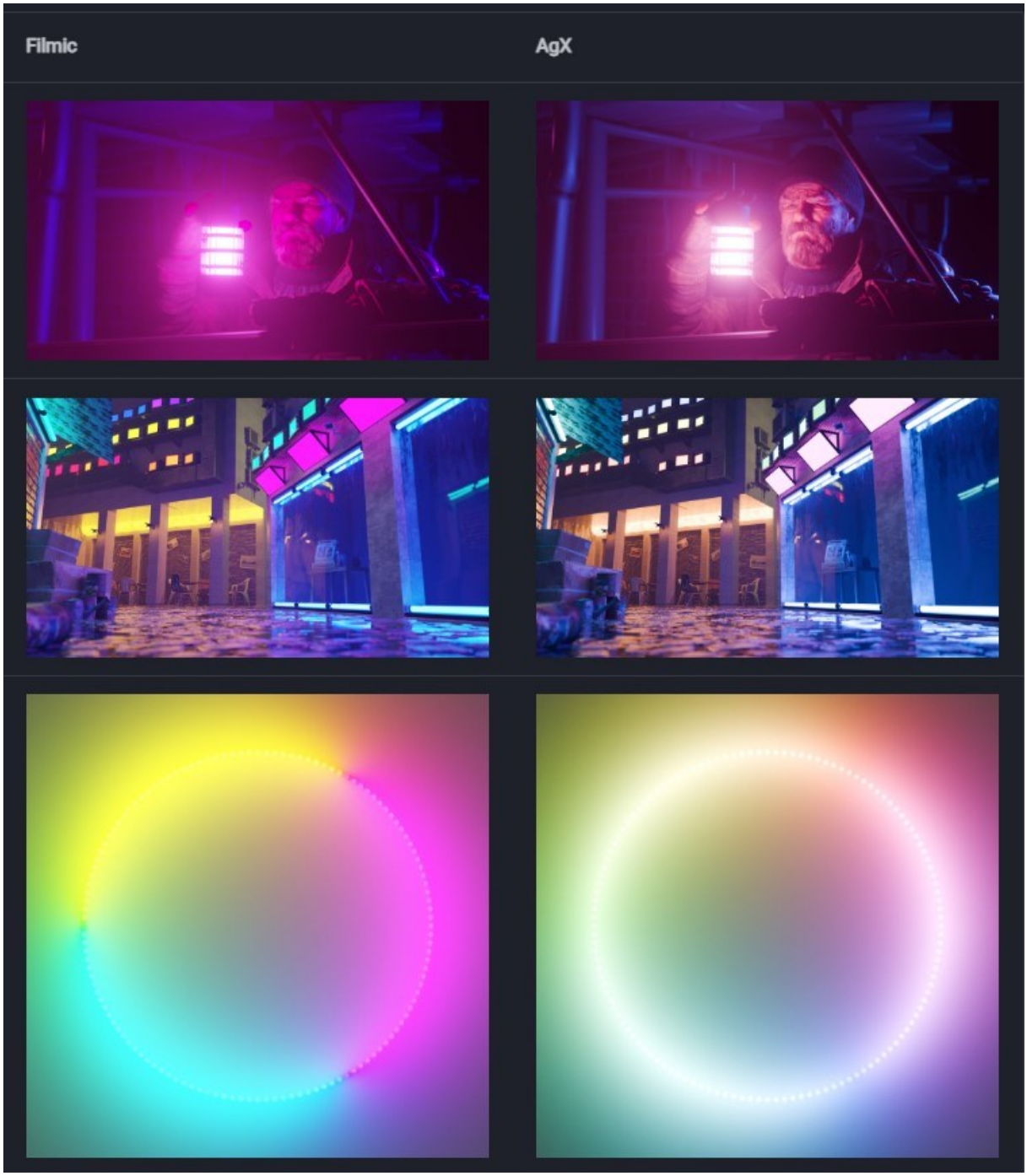
2.5 Post processing

Finally to add the last flavor in realism after the geometry, materials and lighting are rendered. The end result can then be further manipulated by layering a post process filter on top of the rendered image. This can for example be either motion blurring to make low frame-rate appear smoother or a bloom to make a bright light to disperse among the virtual camera's lens creating a sense of blinding light that smudges the vision. Additionally it makes light appear to bleed among the lens, creating flare on the lens to make a sensation that the camera is real. The colors of the rendered image can also be manipulated. For example the rendered frame can be flooded with warm hues of the golden hour before sunset or given cold colors to enhance the shivering wilderness of deserted winter woods. Many post-processing filters can take advantage of the depth buffer and even material surface data to, for example, take roughness into account.

2.6 Color space conversion

According to Gravesen (2015, p. 2) and Blender foundation (n.d.-a), color space conversion or color transform is a method of transforming color data displayed on screen to be more in line of how human perception and real cameras work. Figure 9 shows the difference between Filmic and AgX color transform. There are for example Aces, Filmic and AgX color transforms.

Figure 9. Difference between Filmic and AgX color space conversion. (Blender foundation, n.d.-b)



3 Practical section: visualization of a luxury house

The practical part of this thesis was to make as life-like visualization of a modern house with as realistic visuals as possible. It uses the methods mentioned above to create as realistic as possible modern house tour in unreal engine while reaching towards realism, while diving in with some of the engine's features and settings.

3.1 Software used

The tools used were Unreal engine 5 alonside Blender. They feature wide range of features for photorealistic needs and are gaining more popularity each year. Choosing them was also part of the fact that I used both of these softwares as a hobby and later blender at work as well.

3.1.1 Unreal engine 5

Unreal engine is one of the biggest game engines in the industry. Unreal engine offers tools for various needs in game development ranging from C++ scripting, visual scripting using unreal engine's own proptietary blueprint node based scripting, User interface tools, virtual worlds building with terrain tools, Dynamic lighting with global illumination and many more. It also fully supports ray-tracing and path-tracing. One of the landmark features are Nanite and Lumen. Nanite allows virtually unlimited geometry to be rendered in realtime and Lumen calculated indirect lighting from mesh distance fields. (Epic games, n.d.-d)

3.1.2 Blender

Blender is an open source 3D modeling, rendering and animation software. It was originally developed by a Dutch self-taught software engineer Tom Rosendal in the year of 1994.

Rosendal continued to develop blender for years until 2002 he founded a non-profit organisation named blender foundation and made blender the open source software that is free to everyone. Since then blender has gained major popularity over the years. (Blender Foundation, n.d.-c)

3.2 Beginning

The project was done using Blender first to create some assets and then the unreal engine 5.3. Not everything mentioned were fully done in order presented but roughly in that order.

The practical project started by searching for a ground plan for a modern luxury house. Next it was added to blender for modeling from the ground plan into a fully fledged modern luxury house.

Alongside the house, some furniture objects were also created including speakers. They were modelled as detailed high poly objects while keeping good topology in mind. Figure 10 shows the speakers' topology with wireframe. While Nanite could help run the high poly model in Unreal Engine without much performance impact, low poly model of the same speakers were done for the sake of demonstrating traditional baked normal map workflow for asset creations. To move details from high poly model to the low poly one, the normal maps need to be baked using cycles render engine in blender. After making sure Blender render engine is set to use cycles, samples set to one, color transform set to standard and the low poly model sits at the same coordinates as the high poly version nested on each other, normal maps can be baked from render-bake tab choosing normal in bake type drop down menu. Checking selected to active box after selecting first the high poly- and last the low poly model, normal maps can be finally baked to a texture by pressing the bake button. Figure 11 highlights the crucial settings for the normal bake. The result is shown in the theory section 2.2.1 normal map.

Figure 10. High and low poly model of the same speaker.

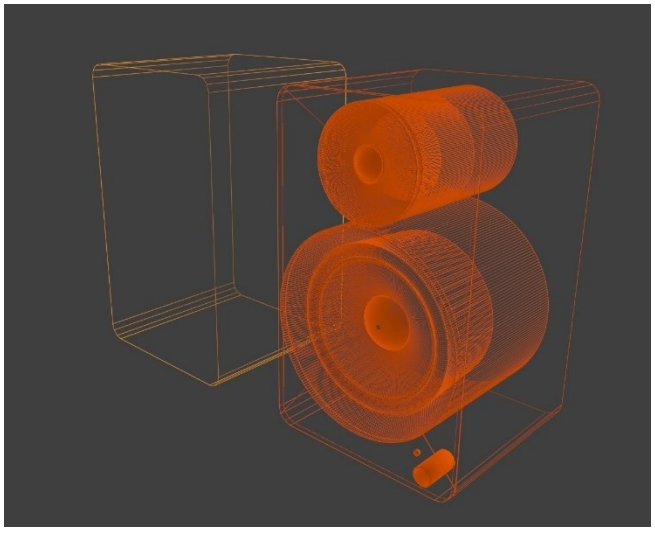
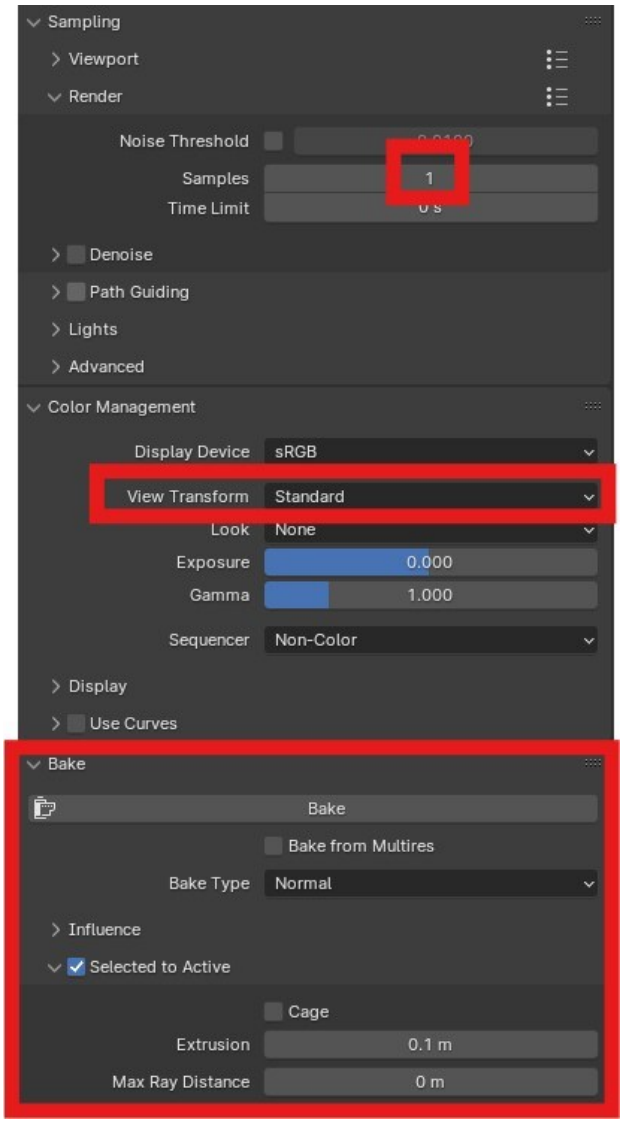


Figure 11. Settings for normal baking.

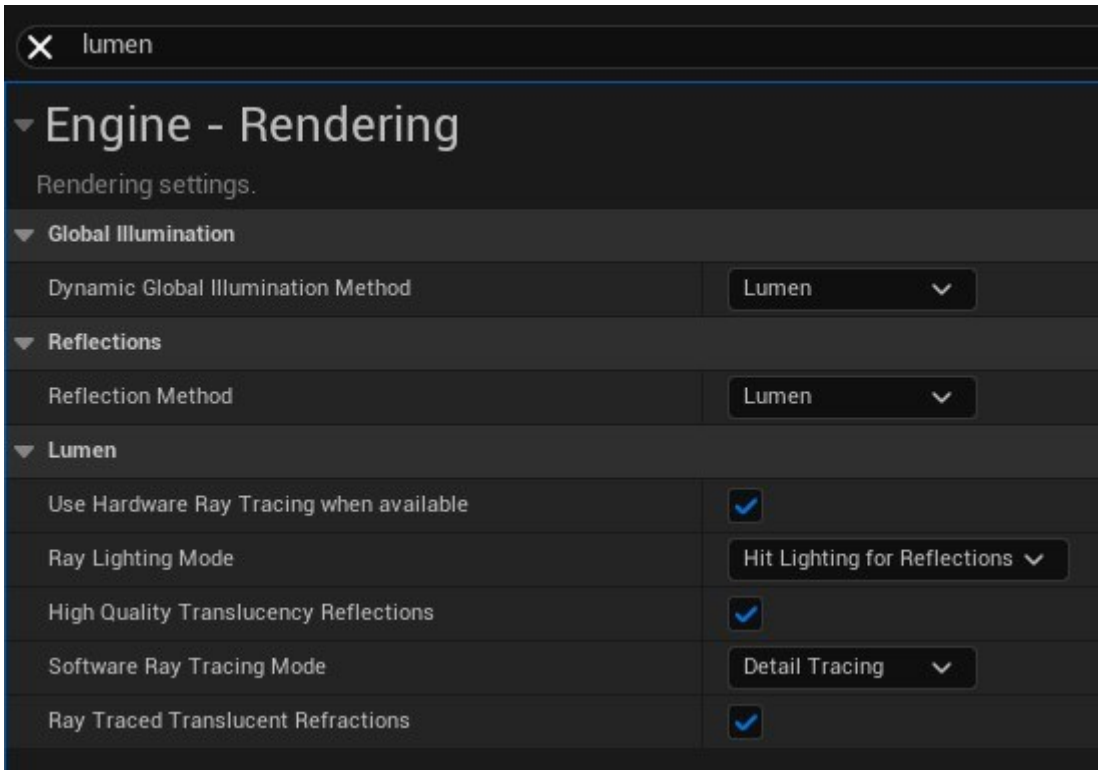


3.3 Setting up lighting

A new blank project for Unreal engine 5 was made. The development started by first adding a lighting setup to a blank new level. Light components used were directional light, exponential height fog, sky atmosphere, sky light with real time capture set to on and volumetric cloud with cloud shadows set to on. Figure 12 shows hierarchy tree's view of the light components. Global illumination and reflections were set to use Lumen from the project settings. Figure 13 shows the Lumen settings used in projects settings.

Figure 12. Components used in lighting up the level.

Figure 13. Lumen settings in project settings.



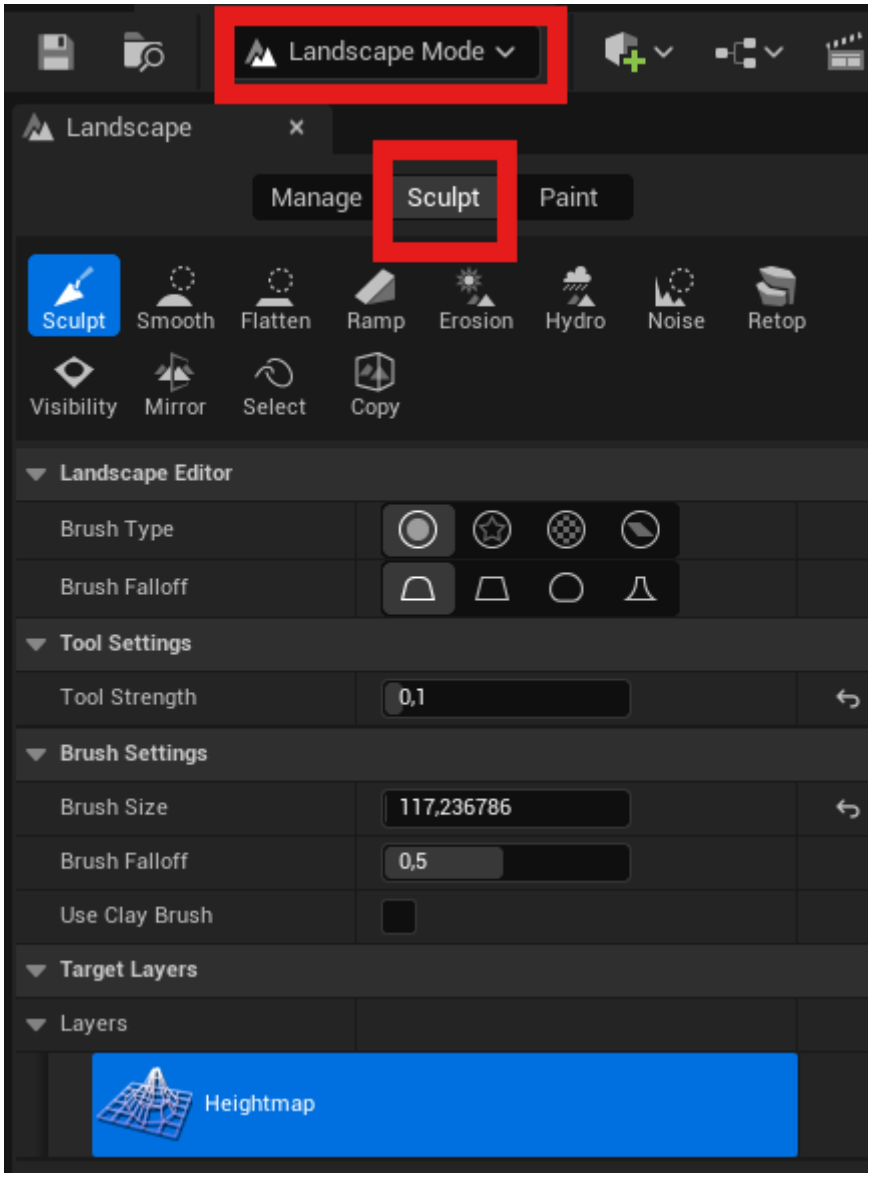
3.4 Scenery

The following step was to create the base ground on which the house stands and the scenery around it. The unreal engine 5 provides great built-in tools for creating beautiful and lush landscapes and scenery. A new landscape was created with the landscape mode activated from the main toolbar and manage tab open from landscape toolbar.

The ground was deflated using the landscape sculpting tool activated from the landscape toolbar. Figure 14 shows Unreal Engine's UI view of landscape sculpt mode. Pit that serves as a great lake for our scenery was made first. A plane was added on top of the pit to act as

a lake. Mountains were created by inflating the ground with the brush. Flat part for the apartment was flattened with flattening brush.

Figure 14. The landscape sculpting tool.



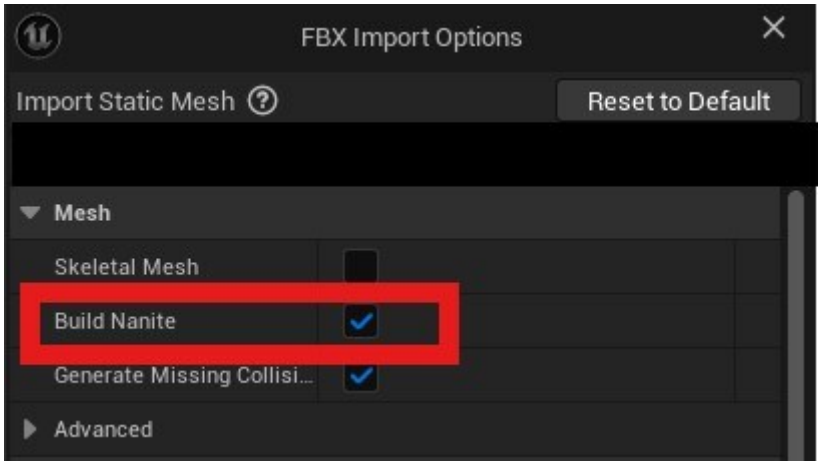
Unreal engine 5's landscape tools provide foliage painting allowing for painting with brush to landscape to add configured foliage assets to painted area. Quixel which is integrated into unreal engine 5, incudes several of these foliage assets to be painted to the landscape. Lush foliage from Quixel were added close to the house with the built in landscape foliage tools.

3.5 Assets

The following assets were made in Blender: The house, kitchen, carpet, poolside, barbeque tent, barstool, living room lamp, wall decoration, TV and speakers were imported into the

scene with build nanite option enabled in the import settings. Figure 15 Highlights the option to build nanite from fbx import settings.

Figure 15. The build nanite option in the import menu.



Shadows from windows were disabled as they caused problems with indoor lighting when using Lumen. Materials were adjusted using Unreal Engine's built-in material nodes. Some translucency was added to the barbeque tent. Some assets were set up using materials from the Quixel's material library. Some of the Quixel materials were edited. The rest of the house and outdoors were populated with assets from blenderkit, which is an asset library addon for blender.

Every object's and light's mobility was set to movable to ensure they use Lumen real time global illumination and reflections so that only Lumen is fully utilised and not light mass light baking. Nanite was enabled for every object except windows.

3.6 Post process

A post-processing volume was added globally, and various effects were used including: Automatic exposure, bloom, lens flare, film grain, vignette, color grading and chromatic aberration. Below is a list of used post processing settings and what has been changed. Everything else not mentioned are default settings.

Bloom: method set to convolution.

Lens flare: Intensity set to 0,1. Bokeh size set to 3. Tints set to use the default tints.

Color grading: Global saturation set to 0,95. Global contrast set to 0,95. Global gamma set to 1,05.

Chromatic aberration: Intensity set to 0,2.

Exposure: Metering mode set to auto exposure histogram. Exposure compensation set to 2.

Film grain: Intensity set to 0,25

Vignette: intensity set to 1.

3.7 Color space conversion

The project was set to use AgX color transform developed Troy Sobotka. It works through OpenColorIO color management system that is integrated into the Unreal Engine 5. As stated earlier it is made to mimic more accurately how a real world camera would work with colors and features. Github user named EaryChow who has made an own branch of the AgX says that: “It provides smooth chromatic attenuation in the image across challenging use cases including wider gamut rendering, real-camera-produced colorimetry etc.” (EaryChow, 2023)

Figures 16 and 17 demonstrate difference between AgX on and off. It gives soft realistic colors that are more natural to the house visualization scene.

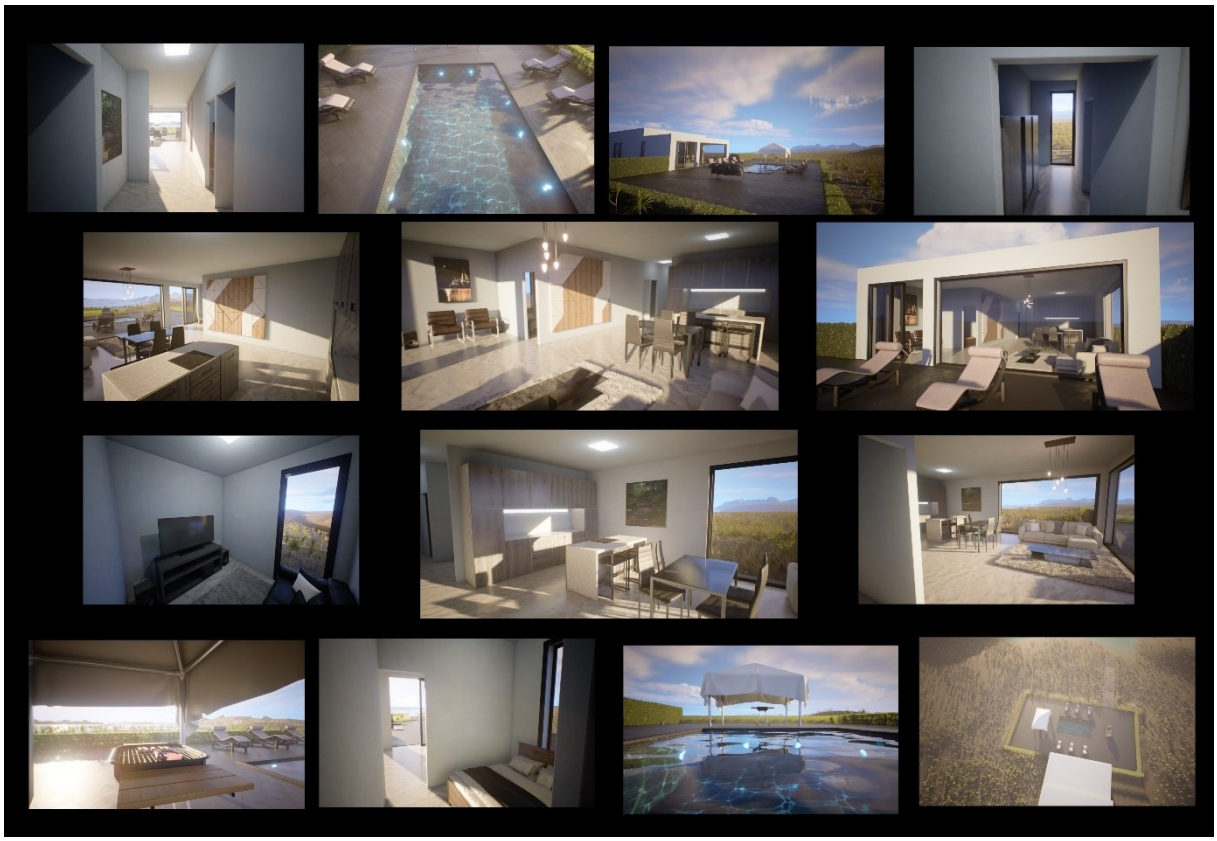
Figure 16. AgX Color space transform: ON

Figure 17. AgX color space transform: OFF

4 Results and discussion

The subject of this thesis was to study what methods could be used to help reach towards photorealism in computer generated real-time graphics and build a visualization based on this knowledge. The methods gathered and explored in the theoretical section helped a lot to achieve these bits of realism. These methods built more understanding and knowledge of how video game graphics work. The results are fairly photorealistic and acts as a great example of how Unreal Engine 5 can be used to make real-time visualizations. Utilizing some of the theory knowledge of the methods gave nice layers of realism in a fairly simple interior scene. At the end of the day it does not have that much detail at all and only a few furniture decorating the interior. This shows that focusing on lighting, post processing, color grading and the materials of the few assets themselves can give great results alone in terms of photorealism. Adding more small details and some scattered everyday objects that would make the interior resemble more like a lived house would have helped to create the scene to come even more believable in terms of realism. Figure 18 shows the final results of the house’s visualization. The theory base that this thesis offers opens up a broad branch of research.

Figure 18. Final results of the visualization.



References

Arm Limited. (2020). *Real-time 3D Art Best Practices – Geometry.* https://developer.arm.com/documentation/102448/0100/?lang=en

Blender Foundation. (2024). *Texture & Texture Mask.* https://docs.blender.org/manual/en/latest/sculpt_paint/brush/texture.html

Blender Foundation. (n.d.-a). *Blender 4.0: Color Management.* https://developer.blender.org/docs/release_notes/4.0/color_management/

Blender Foundation. (n.d.-b). *Blender 4.0: AgX view transform* [Figure 9]. https://developer.blender.org/docs/release_notes/4.0/color_management/

Blender foundation. (n.d.-c). *Blender's history*. https://www.blender.org/about/history/

Dunn, I. & Wood, Z. (n.d.). *Physically-Based Rendering Cook-Torrance Reflectance Model.* https://graphicscompendium.com/gamedev/15-pbr

Epic Games. (2024, May 13). Magnopus brings Amazon's Fallout series to life with virtual production powered by Unreal Engine. *Unreal Engine spotlights.* https://www.unrealengine.com/en-US/spotlights/magnopus-brings-amazon-s-fallout-series-to-life-with-virtual-production-powered-by-unreal-engine

Epic Games. (n.d.-a). *Nanite Virtualized Geometry*. https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine

Epic Games. (n.d.-b). *Lumen Technical Details*. https://dev.epicgames.com/documentation/en-us/unreal-engine/lumen-technical-details-in-unreal-engine

Epic games. (n.d.-c). *Light Propagation Volumes*. https://docs.unrealengine.com/4.27/en-US/BuildingWorlds/LightingAndShadows/LightPropagationVolumes/

Epic games. (n.d.-d). *Unreal Engine 5.3 Documentation*. https://dev.epicgames.com/documentation/en-us/unreal-engine/unreal-engine-5-3-documentation?application_version=5.3

Evanson, N. (2023). *Path Tracing vs. Ray Tracing, Explained*. Techspot. https://www.techspot.com/article/2485-path-tracing-vs-ray-tracing/

Farris, J. (2020, February 20). Forging new paths for filmmakers on "The Mandalorian". *Unreal Engine blog*. https://www.unrealengine.com/en-US/blog/forging-new-paths-for-filmmakers-on-the-mandalorian

Filmmakers academy. (2021). Are game engines the future of filmmaking? *Filmmakers academy*. https://www.filmmakersacademy.com/are-game-engines-the-future-of-filmmaking/

Freudenrich, H. (2013). *How Light Works*. New York University NYU. https://wp.nyu.edu/being_digital/wp-content/uploads/sites/1503/2015/08/Light-Waves.pdf

Geupel, M. (2022). *Albedo and its effect on photorealism*. https://www.racoon-artworks.de/cgbasics/albedoandphotorealism.php

Gravesen, J. (2015). *The Metric of Colour Space*. Technical University of Denmark. http://www2.mat.dtu.dk/people/J.Gravesen/pub/48-2015-colour.pdf

Heginbotham, C. (n.d.). *What is 3D Digital Sculpting?* Concept Art Empire. https://conceptartempire.com/what-is-3d-sculpting/

Hukseflux. (n.d.). *What is albedo?* https://www.hukseflux.com/library/what-is-albedo

Kajiya, J. (1986). The rendering equation. *ACM SIGGRAPH Computer Graphics*, 20(4), 143–150. https://www.cse.chalmers.se/edu/year/2011/course/TDA361/2007/rend_eq.pdf

Karis, B. (2013). *Real Shading in Unreal Engine 4.* Unreal Engine. https://cdn2.unrealengine.com/Resources/files/2013SiggraphPresentationsNotes-26915738.pdf

Lambru, C., Morar, A., Moldoveanu, F., Asavei, V. & Moldoveanu, A. (2021). Comparative Analysis of Real-Time Global Illumination Techniques in Current Game Engines. *IEEEAccess, 9,* 125158 – 125183. https://doi.org/10.1109/ACCESS.2021.3109663

Learn OpenGL. (n.d.). Ssao. https://learnopengl.com/Advanced-Lighting/SSAO

Linietsky, J., & Manzur, A. (n.d.). *Signed distance field global illumination. (SDFGI).* https://docs.godotengine.org/en/stable/tutorials/3d/global_illumination/using_sdfgi.html

Marsh, A. (2010). *Specularity*. https://performativedesign.com/definition/light-specularity/

Math worksheet center. (n.d.). *Video Game Graphics: It's all about Geometry!* Math worksheet center. https://www.mathworksheetscenter.com/mathtips/videogamemath.html

Martin, J. (2021). Modeling with animation in mind. https://topologyguides.com/modeling-for-animation

McCarney, J. (2018). Topology research [figure 1]. *Jack's blog.* https://kcddjackmccarney.wordpress.com/2018/09/12/topology-research/

Morgenstern, Y., Hartmann, F., Schmidt, F., Tiedemann, H., Prokott, E. & Maiello, G. (2021). An image-computable model of human visual shape similarity. *PLoS Comput Biol, 17*(6), 1 – 34. https://doi.org/10.1371/journal.pcbi.1008981

Nanda, V. (2023). *Difference between Albedo and Reflectance.* https://www.tutorialspoint.com/difference-between-albedo-and-reflectance

Schenk, T. (2005). *Introduction to Photogrammetry.* The Ohio State University https://www.mat.uc.pt/~gil/downloads/IntroPhoto.pdf

Unity technologies. (n.d.). *Light Probes*. https://docs.unity3d.com/Manual/LightProbes.html

Vries, J. (n.d.). *PBR Theory*. https://learnopengl.com/PBR/Theory

Weisstein, E. (n.d.). *Albedo*. https://scienceworld.wolfram.com/physics/Albedo.html

Whitted, T. (2018). A Ray-Tracing Pioneer Explains How He Stumbled into Global Illumination. *Nvidia blog.* https://blogs.nvidia.com/blog/ray-tracing-global-illumination-turner-whitted/

Wright, D., Narkowicz, K. & Kelly, P. (2022). *Lumen Real-time Global Illumination in Unreal Engine 5.* Siggraph 2022. https://advances.realtimerendering.com/s2022/SIGGRAPH2022-Advances-Lumen-Wright%20et%20al.pdf

Williams, L. (n.d.). *Casting curved shadows on curved surfaces.* New York Institute of Technology. https://cseweb.ucsd.edu/~ravir/274/15/papers/p270-williams.pdf

Appendix 1. Photorealistic real-time visualization of the modern luxury house.