

Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Proyecto Final

Custom Raytracing Material Library for games

Autor del proyecto: Francisco Núñez Villagómez
Director del proyecto: Eduardo Jiménez Chapresto
Zaragoza, 7 de septiembre de 2022



Este trabajo constituye parte de mi candidatura para la obtención del título de Graduado en Ingeniería Informática por la Universidad San Jorge y no ha sido entregado previamente (o simultáneamente) para la obtención de cualquier otro título.

Este documento es el resultado de mi propio trabajo, excepto donde de otra manera esté indicado y referido.

Doy mi consentimiento para que se archive este trabajo en la biblioteca universitaria de Universidad San Jorge, donde se puede facilitar su consulta.

Firma

Fecha

Dedicatoria y Agradecimiento

Me gustaría agradecer este trabajo a Eduardo Jiménez por su orientación y dedicación no solo durante este proyecto sino desde el inicio de la carrera.

Asimismo, quiero dar las gracias a mis compañeros, en especial a Gabriel Villalonga, Daniel Gracia y Jorge Solano por darme estos años juntos y compartir mi pasión.

También quiero realizar una mención especial a los profesores en general, y en especial, a Jorge Echeverría, por confiar en mí y darme todo por conseguirme oportunidades para desarrollar mi potencial.

Por último, quiero agradecer a mis familiares más cercanos por apoyarme en mis decisiones y hacer todo lo posible por que pudiese perseguir mis objetivos.



Tabla de contenido

Resumen	1
Abstract.....	2
1. Introduction	3
1.1. The role of the Technical Artist	3
1.2. Visual Development of Materials in Games.....	4
1.3. Materials in Unreal Engine 5	4
2. State of the Art.....	8
2.1. A brief history of Videogame Art.....	8
2.1.1. The Evolution of Graphics in Games.	10
2.1.1.1. Text-based graphics.....	10
2.1.1.2. 2D Vector Graphics	12
2.1.1.3. 2D Raster Graphics	13
2.1.1.4. 3D Graphics.....	18
2.1.1.4.1. Texture Mapping	19
2.1.1.4.2. Lighting	21
2.2. Recent History.....	22
2.2.1. Physically based Rendering Materials.....	22
2.2.2. Real time Raytracing	23
2.3. Chess games graphics	25
2.4. Raytracing Material Library	28
3. Objectives.....	30
4. Methodologies	31
4.1. Tools.....	31
5. Development	35
5.1. Raytracing Research and Development in Unity	35
5.2. Available Technology, why Unreal Engine 5.	37
5.3. Master Material and Material Instances	38
5.4. Material Layers and Material Layer blends	39
5.5. Material Functions.....	39
5.6. Plastic materials	40
5.7. Glass materials.....	44
5.8. Water materials.....	46
5.9. Dust/Frost	49
5.10. Post-process volume sublevels	51
6. Economic Study	55
7. Results.....	57
8. Conclusion and future work	61
9. Bibliography	63
10. Annexes	65
10.1. Project Proposal	65



10.2. Meetings.....	66
----------------------------	-----------

Resumen

El Raytracing es una técnica de renderizado que permite conseguir imágenes con efectos de luz extremadamente realistas. Hasta hace poco esta técnica no era posible realizarla en tiempo real debido a su alto coste computacional. Sin embargo, con la última tecnología desarrollada por las empresas productoras de GPUs, se está pudiendo lograr implementar sistemas que aprovechen el Raytracing en tiempo real.

Esta técnica busca simular mediante un algoritmo el camino que sigue un rayo de luz, y simular la manera en la que este rayo interactúa con objetos virtuales representados en el mundo 3D generado. Dicha simulación permite a desarrolladores representar sombras, refracciones y reflejos de una manera mucho más parecida a como el ojo humano lo haría.

Anteriormente, el Raytracing ha sido usado en el campo del renderizado de imágenes digitales, tales como películas y programas de televisión. Actualmente se está comenzando a utilizar en videojuegos de maneras puntuales, pero en constante aumento.

En este trabajo se incluyen diferentes ejemplos de materiales que usan técnicas tradicionales en conjunción con técnicas que emplean Raytracing para conseguir imágenes más realistas y su aplicación en videojuegos.

Abstract

Raytracing is a rendering technique used to produce high quality and realistic light effects. Not a long time ago this technique was not possible to be used efficiently in real time practices due to its high computational cost. Thank to recent GPU technology advancements, it is possible to create systems that take advantage of real time Raytracing.

This technique simulates through algorithms the path a light ray follows, and all the interactions that ray would have with the virtual objects that are represented in the 3D generated world inside the computer. Said simulation allow developers to represent shadows, refractions and reflections in a way that is much more similar to how the human eye would perceive them.

Recently, Raytracing was used in the image rendering field to generate images for films or TV shows. Nowadays, videogames are starting to use this technique in some ways in constant evolution.

In this Project are included different examples of materials that use traditional techniques in conjunction with Raytracing features in order to achieve more realistic images for videogames.

1. Introduction

My main responsibility in this project is to develop the Material Library that is going to be used in the Magic Chess Eclipse Games game project. My principal focus will be on Raytracing capable Materials, but I will also develop other kinds of materials.

As I entered the development really early, changes happened constantly so I had to overcome different issues that arose during the process and find solutions to those problems in a way that would be beneficial to the project. Some of these issues were: changing the game engine, adapting previous materials, and learning new technologies as Epic games released them into Unreal Engine 5.

A Technical Artist in videogames most of the time mixes art and computer graphics techniques to develop assets that can be used by other members of the team to produce a better product. My system is composed of different kind of materials that tries to mimic real life materials, so I had to constantly tweak values and test it to achieve the desired result. A good thing about this is that, as materials are specific kind of assets, they can be decoupled from other systems and introduce more changes without disturbing other members of the team.

The Magic Chess project is Eclipse Games take on the old Battle Chess and adapting that gameplay to today's systems. That would turn Magic Chess into a Chess Strategy Game with the gimmick that the pawns, rooks, kings, and other pieces move and attack visually. It has a small touch from War Games in the way that different armies are depicted, and you can confront them, without losing the essence of chess. As previously said, my work in this project will be focused on the materials.

1.1. The role of the Technical Artist

It has been said that the Technical Artist is the connection between the designers and artists, and the engineers, as the profile of technical artist is of someone capable of having a deep understanding of the technical requirements of Real time rendering, the different rendering pipelines that exist or are being used within the project, and the performance capabilities of the different target platforms where the product is going to be shipped on.

A Technical Artist is typically part of the Research and Development Team, where other professionals with different backgrounds help each other develop features that will help different fields. From tools that improve workflows to enhancing existing assets of the project. As an example, in a Riot Games documentary, a Technical Artist was working on a tool that helped animators that were working on rigging to have better weights in the different bones, allowing for a smooth and more realistic animation which didn't deform polygons in a way that was undesirable.

A technical artist can also work on developing and enhancing materials to fit the performance and visual needs of a project, from adding new techniques within the material to improve visuals, to reducing the size of textures or compressing data in a way that don't change the visual appearance of the material but improve the memory usage, resulting in a performance gain. Combining different types of assets and using different techniques such as data compression and reinterpretation, technical artists can create new assets that with a bit of value tweaking can

range from toon to real life material models. As we can see from the information above, this kind of role needs to work closely with both designers, artists, and engineers to turn their needs into a reality.

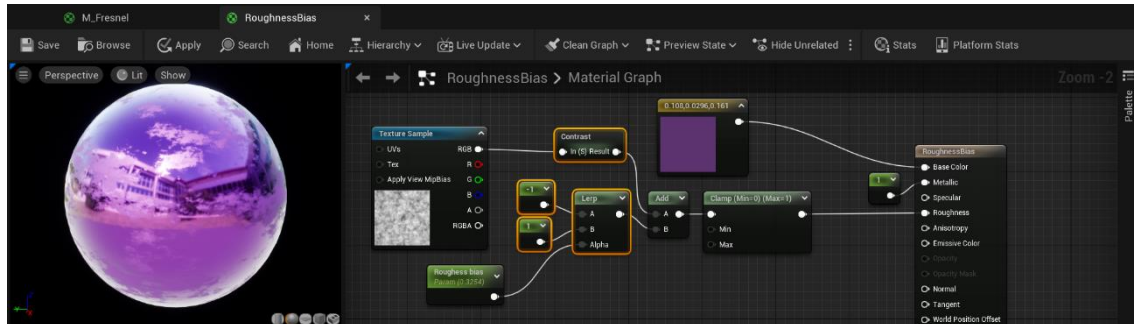


Figure 1: Unreal Engine 5 Material graph system

Being able to understand systems featuring particles, shaders, materials, textures, post processing effects and being able to optimize them is a huge plus to the background of a technical artist, and almost every technical game developing job type of background. Successfully transmitting these features into a game is a huge part of the job of a technical artist.

1.2. Visual Development of Materials in Games

Material and visual development in games are related terms most of the time. Independently of the visual aesthetic of our game, materials and 3D models will be the most important assets that will help achieve the desired visual quality.

In our case, we want to use Physically Based Materials (PBR), which we will talk about in the next chapters, as well as make use of the Raytracing capabilities of Unreal Engine 5 to bump up visuals and create realistic lighting in the Magic Chess project scenes.

The material library system will provide the desired functionality and capabilities to bring Magic Chess visuals to life and also make our designers and artists job easier due to the fast variation of the same material, allowing for visual diversity in scenes, as we will see in the development section of this document.

1.3. Materials in Unreal Engine 5

Materials are the Rendering solution to define the different surface properties of objects in a scene. As such, different materials will have different properties and maybe the same material might have multiple instances of it with distinct values to it. Materials also define how a surface interacts with the light in all aspects of the surface: color, reflection, refraction, bumpiness, transparency...

Materials can range from cartoony to Physically Based Materials, which try to mimic real life surfaces by using physically accurate models or estimations taken from real life.

Unreal Engine 4 and 5 use a Material graph, which is a system that allows designers and Technical Artists to work directly on the materials and provide a rendered version of it in real time. It allows the person who uses it to do several things:

- Create Material Functions that can be reused in all materials.
- Layers, which allow to add different effects to materials such as a glass with a layer of frost on top to simulate a very cold environment, or dust to simulate a dirty place.
- Also, the same Material can be recreated by using Material Instances, which allow to input different values to the same material in order to generate variations from it.

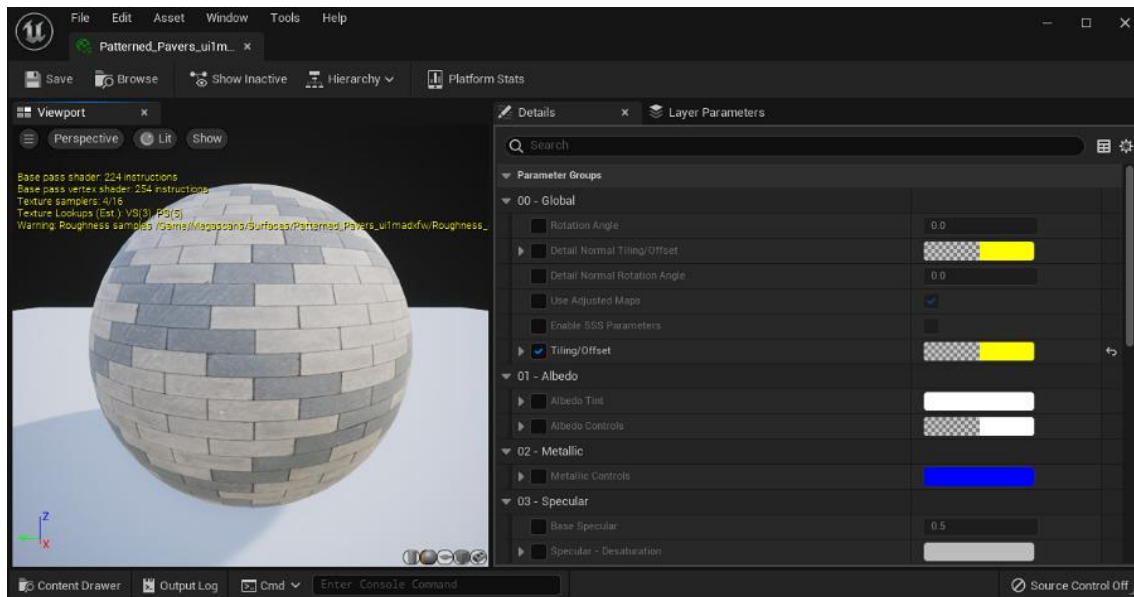


Figure 2: Unreal Engine 5 Material Instance view

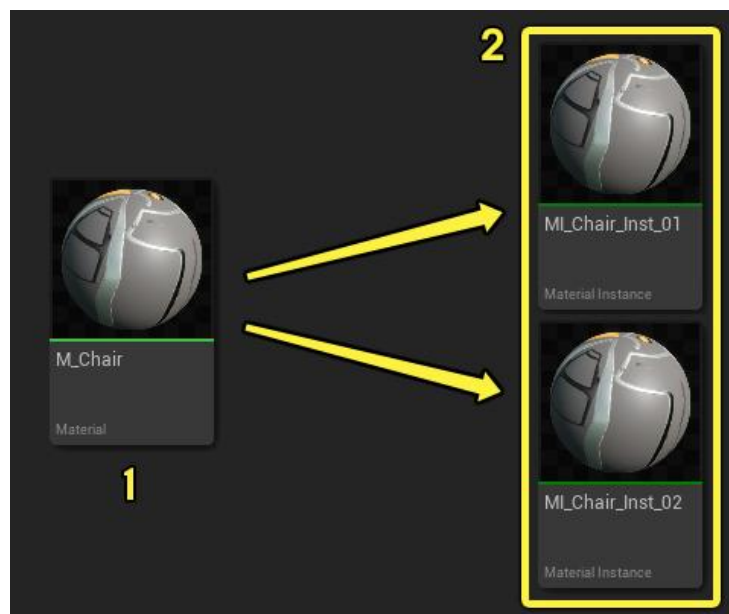


Figure 3 Material Inheritance in Unreal Engine 5

Materials are a visual way of adding effects and features to the rendering of the project, some effects that can be achieved are normal mapping, custom UV mapping, refraction effects, screen space effects, tessellation...

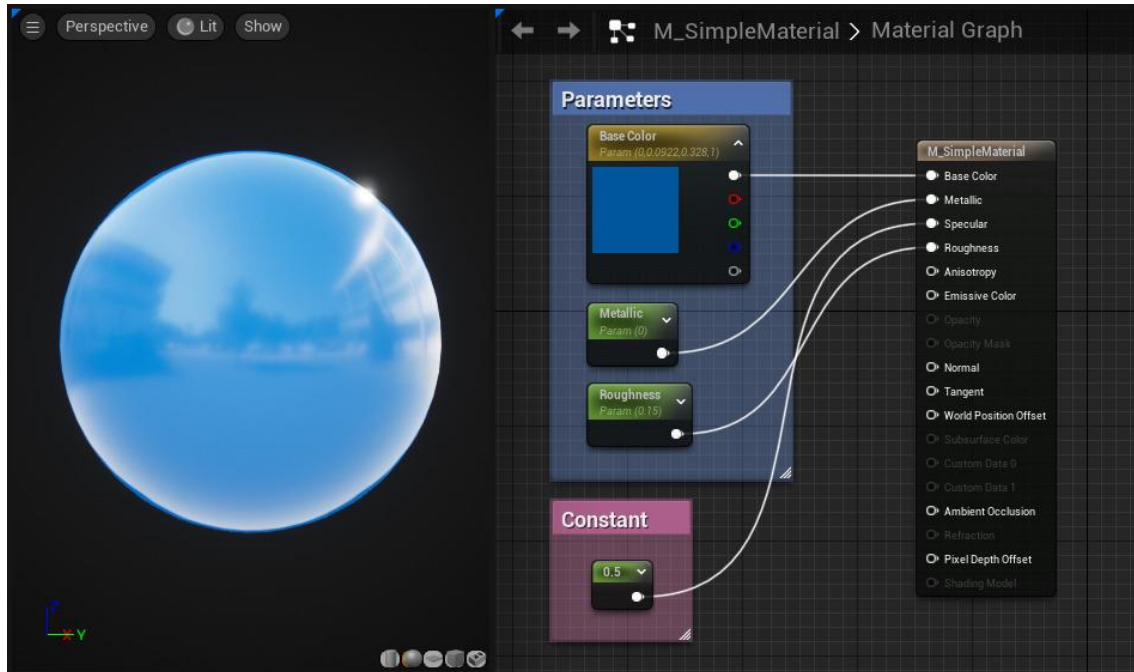


Figure 4 Unreal Engine 5 Material Example depicting parameters

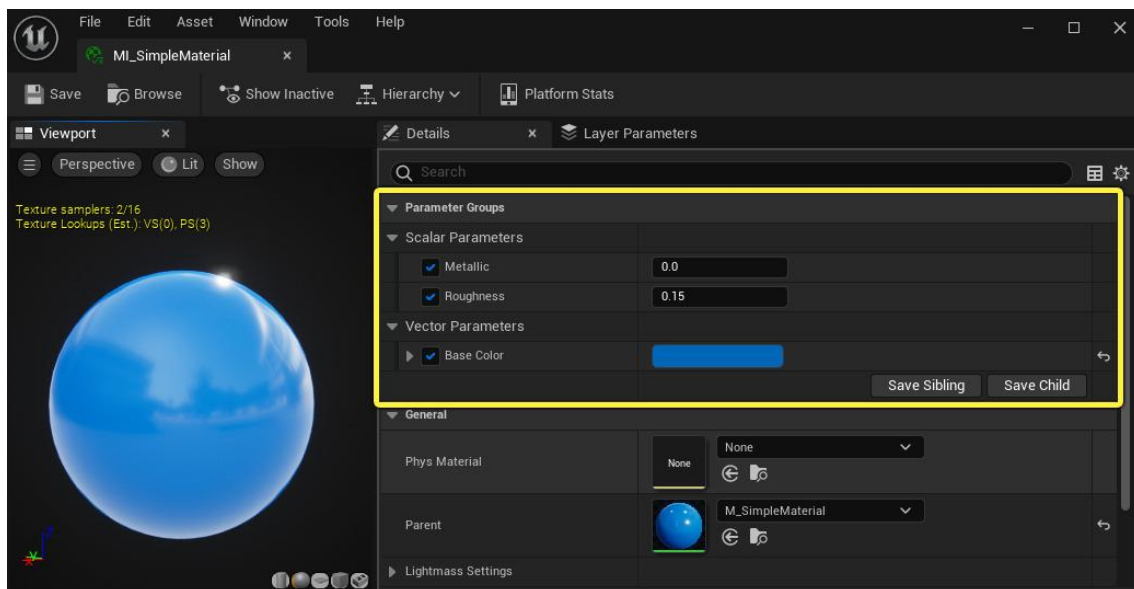


Figure 5: Unreal Engine 5 Material Instance from previous material

As we can see in the previous figures, the material and material instances system are very flexible and can be iterated upon very easily. I used this system in Magic Chess as it allowed me to create different variations of the same material which is very helpful to iterate and generate a visual difference between objects in the game, which is something appealing.



2. State of the Art

Game graphics weren't always so impressive as they are today. However, every time a major graphics improvement happened it was welcomed and cheered. As an example, at the very beginning, games didn't even showcase images, but text.

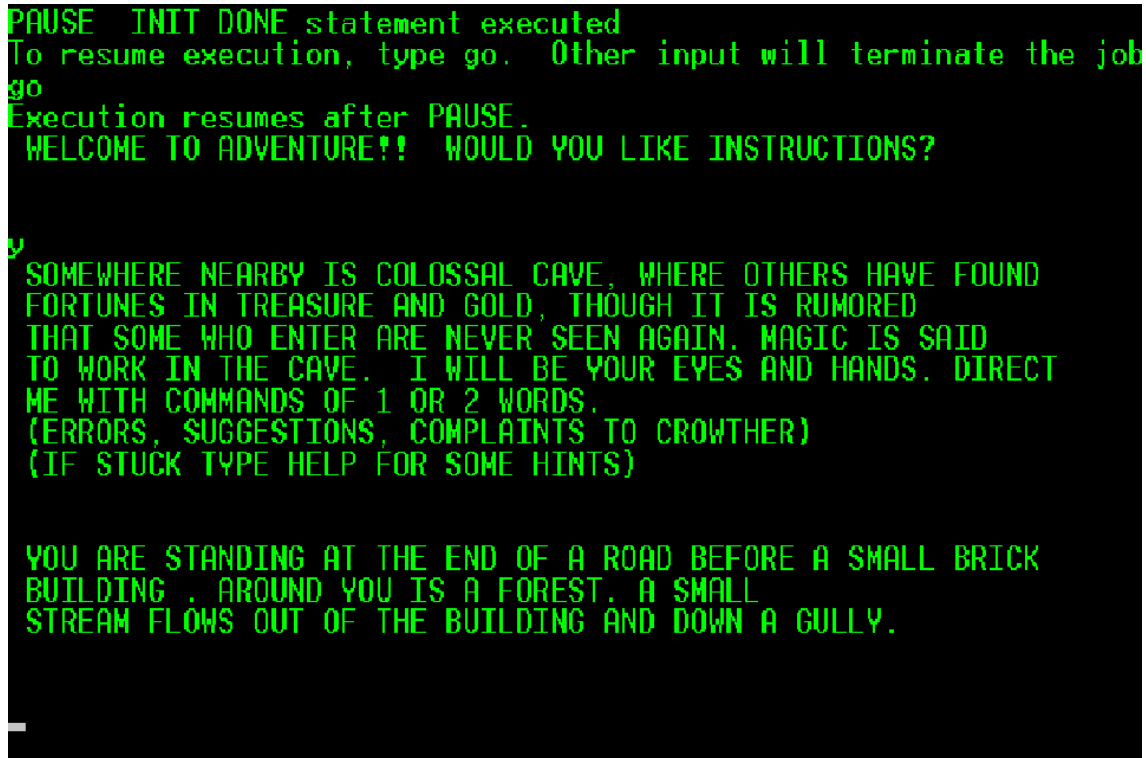


Figure 6: Colossal Cave Adventure, a text-based game.

Innovation in graphics didn't come all together, but in small pieces and featuring different algorithms which tried their best to do what was needed at the time. As hardware started being more powerful and started consolidating, graphics started being more flamboyant and different advanced techniques were implemented.

As graphics started being more and more important for game development, different roles started taking part in the development: 2D and 3D Artists, Graphics/Rendering Engineers, Animators... and Technical Artists.

2.1. A brief history of Videogame Art

As materials are a huge part of videogame art and the main focus of this project, we will take a look at how videogame graphics and art turned out to be how it is nowadays.

Investigating about how videogame art technology evolved allowed me to think more about how to implement the different materials that are featured within the Magic Chess project. Also, this knowledge will help understand the different decisions that were made along the way by game developers to reach how graphics work these days.



Figure 7: Tomb Raider game protagonist, Lara Croft, graphics development

For this research I divided the information in several chapters, depicting the evolution of the different fields within graphics programming and specially taking a better look at games which would be of interest for the development of the Magic Chess project such as Battle Chess, Chessmaster 3D, Battle Vs. Chess and Chess Ultra.



Figure 8: Chess Ultra screenshot

2.1.1. The Evolution of Graphics in Games.

Videogames have been and are one of the engines that impules innovation in real time graphics rendering. From showing sprites and animating them in a pixel grid to showcasing millions of polygons or using information gathered through raytracing techniques to compose a high-end realistic image.



Figure 9: Evolution of game characters graphics featuring Luigi, Mario and Megaman

All done in less than 33.3 milliseconds for 30 frames per second, or 16.7 milliseconds for 60 frames per second. To achieve the level of quality we enjoy today, both hardware and software have had to improve substantially, in the next chapters we will feature the most important advances that each field of graphics has had. Starting from Text-based graphics and passing through Vector graphics, 2D graphics, 3D graphics, Physically Based Rendering Materials, to Raytracing.

2.1.1.1. Text-based graphics

This kind of graphics aren't exactly graphics, but text. This technique is also referred to as "character based" and most of the time only features characters, however sometimes it could draw little illustrations by using special fonts that allowed to do that: the OEM character set, which contained line draw symbols to create charts and very simplistic graphics.

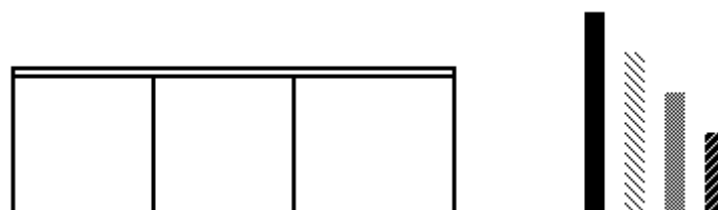


Figure 10: OEM Font line draw examples.

This kind of technology was widely adopted by role playing games or text adventures such as Colossal Cave Adventure, Zork and Hitchhiker's Guide to the Galaxy, where the player would read depictions of rooms, enemies, items, NPCs, or events that happened and had some options on what to do such as crossing doors to new places, attack enemies, pickup items, use them..., There were also text-based online virtual worlds originally called Multi-User Dungeons, where different users could chat, read depictions of rooms, objects and other players, and perform different actions depending of the game. Other of the most prominent genres was Roguelikes, which were derived from the role-playing games, but focused on the replayability, permanent death and turn based type of movement in a dungeon crawl environment. This last genre used many keys as input to extend the interaction with items and environment. Its name was minted by the game called Rogue.

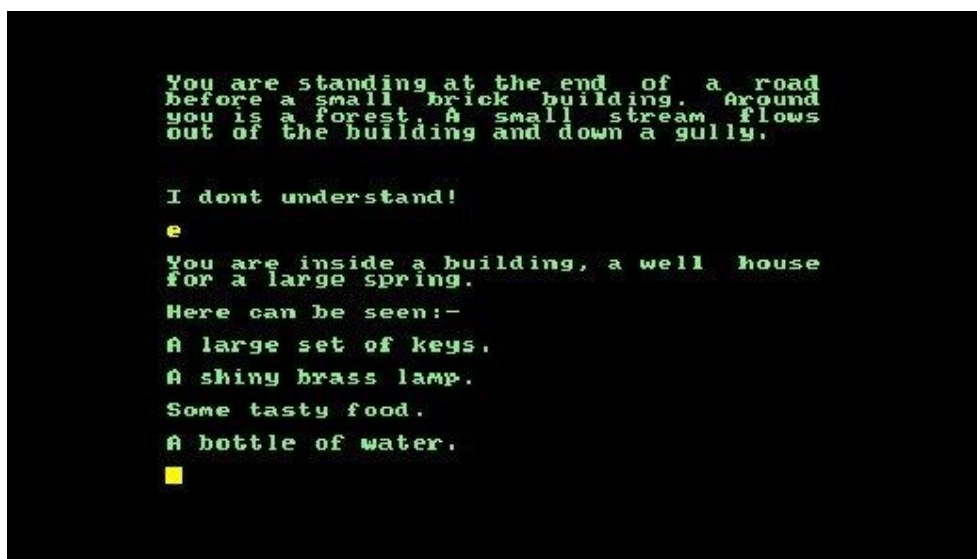


Figure 11: Colossal Cave Adventure screenshot

Even though this technology can look rudimentary, impressive games have been done using it, and some are still on development. There is one example in particular: Dwarf Fortress, which combines text-based graphics with procedural generation of almost everything imaginable from world, terrain, characters, events, enemies, items, nations, conflicts to even letting the player build their own civilization from scratch. It is considered the most complex game ever made in terms of learning how to play it due to the high number of systems that work underneath and the especially complex controls to play.

But not everything is role playing games or words depicting things, there are also chess games developed with this aesthetic, as we can see in the figure at the right.

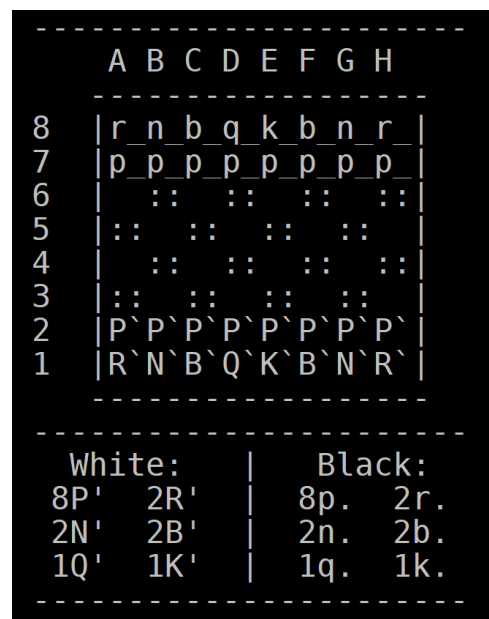


Figure 12: Text-based chess game by bmeares on github.com

2.1.1.2. 2D Vector Graphics

This kind of graphics used mathematical formulas to create different shapes. Vector graphics are 2D lines that connect different points. The first use case of this technology was in oscilloscopes used as displays in the 50s. Due to the reduced memory of these devices, displaying raster or bitmap images was not possible. The Whirlwind computer was the first to display images in modified oscilloscopes by using vector graphics.

The first time this technology was used in game systems was by the hand of the Vectrex home gaming system, which led to the creation of different arcade games such as Asteroids, Space wars as well as some cinematics such as Rip-Off.

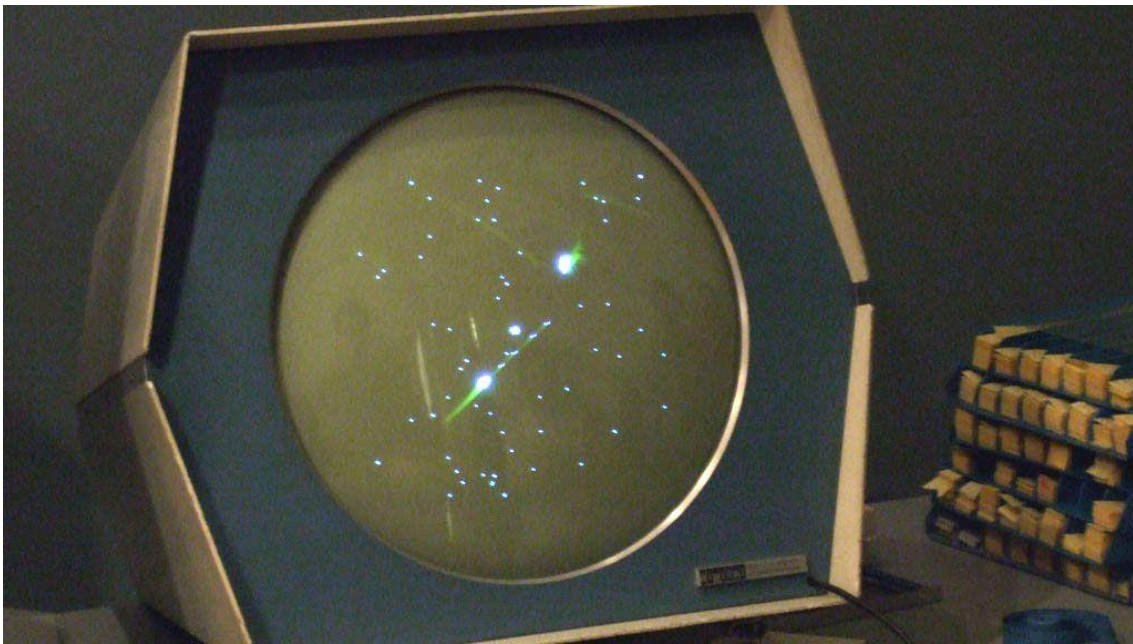


Figure 13: Space war played in an oscilloscope display

One of the most important steps in the vector graphics evolution and in game graphics was the introduction of Bézier curves, which are parametric curves defined by a set of discrete points called control points. This method made possible to approximate real life shapes that had no mathematical expression, or a very complex one. Later on, these curves led to the implementations of Bézier splines, which are the generalization of the curves to higher dimension systems. These curves didn't stay on vector graphics only, they are widely used in game development and graphics to create smooth animations, model terrain in semi-procedural generation tools, visual effects, and particle physics.

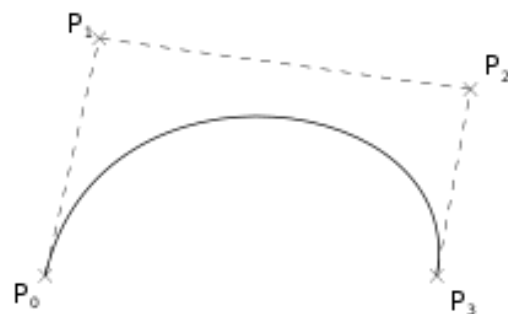


Figure 14: Bezier Curve visual representation

One of the most impressive games done with vector graphics is Star Wars (1983), a first-person rail shooter which was initially an arcade game developed by Atari. It featured impressive color vector graphics which used a projection algorithm to simulate a 3D look, at that time this game was using cutting edge technology. Along with Starglider, it is thought to be the precursor of the Star Fox games.



Figure 15: Arcade Star Wars game

Vector graphics in game started to decline due to the improvements in 2D sprite rendering technology and rasterization of 3D filled polygon graphics.

2.1.1.3. 2D Raster Graphics

For a very long time, 2D Raster Graphics were the most popular kind of graphics and lots of games were developed using them. Entertainment systems started having 2D graphics chip/graphics processing units from the 8-bit, 16-bit, arcade systems where they needed to support multiple sprites on the display.

These graphics drew to an array of pixels also known as bitmap, which was filled with a background uniform color. The drawing was made in an ordered way, placing patterns of color to generate different forms. This canvas could be the framebuffer of a computer display.

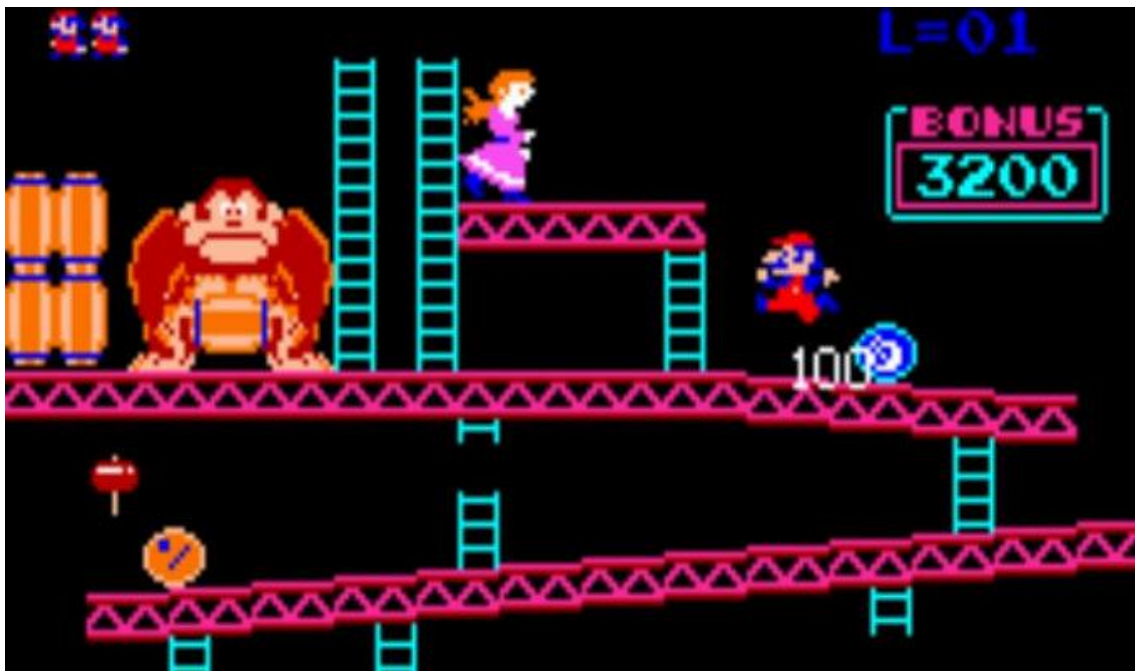


Figure 16: Donkey Kong 1981

At the very beginning of this kind of graphics, the pixels were drawn directly. However, as time passed, game developers started relying on 2D graphics library or graphic cards which would implement features such as:

- Pasting an image at a given position in the canvas.
- Write character strings with a specific font, given a position and an angle.
- Paint simple geometric shapes such as triangles, circles, squares...
- Draw lines, arcs, curves...

Having this basic functionality, graphics programmers could implement more complex systems such as sprite sheets, where sprite strips were loaded and drawn on top of each other at different frames to generate 2D animation.

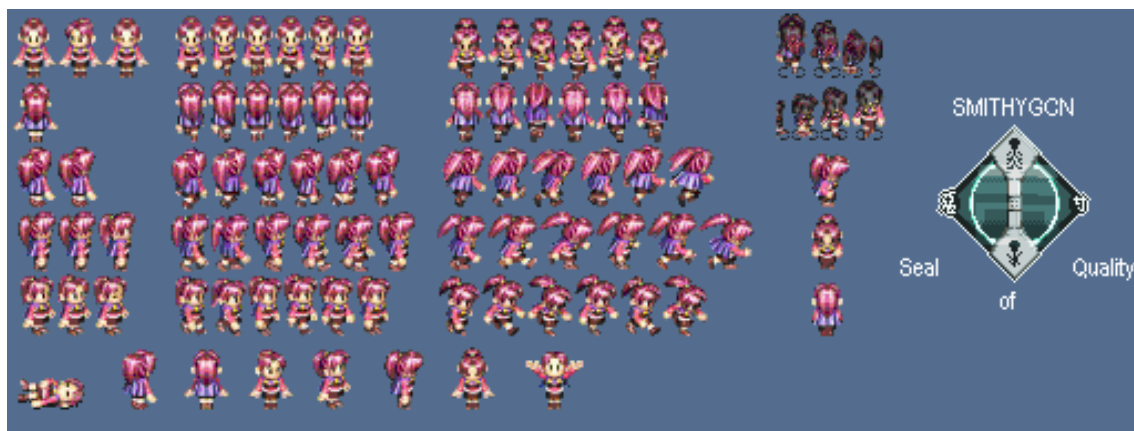


Figure 17: Golden Sun I Sprite Sheet

As 2D graphics systems didn't provide three dimensional shapes or optical features, there were used some techniques to simulate them. One of them is having different layers in front of each other, creating a depth illusion. Layered models are called 2.5D graphics, and different effects were created from this.

Parallax scrolling is one of the most known effects derived from layering, where different background images stacked in front of each other move at different speeds to generate a depth illusion.

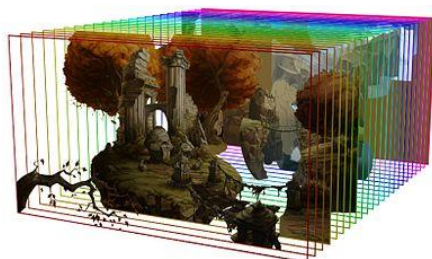


Figure 18: The Whispered World Parallax



Figure 19: The Whispered World Parallax from the front

2.5D or pseudo-3D perspectives were generally done to simulate the appearance of a three dimensional space in a system that was incapable of rendering it. Some techniques to get this look are:

- Axonometric projection, which is a type of orthographic projection where objects are rotated to reveal multiple sides. This projection can be divided taking into account the measures of the sides: Isometric, when all sides are of equal measure, Dimetric, when two of the sides are symmetrical and a third is unsymmetrical, and finally trimetric, where all sides are unsymmetrical. The most commonly used in games is Dimetric with a 2:1 pixel ratio.

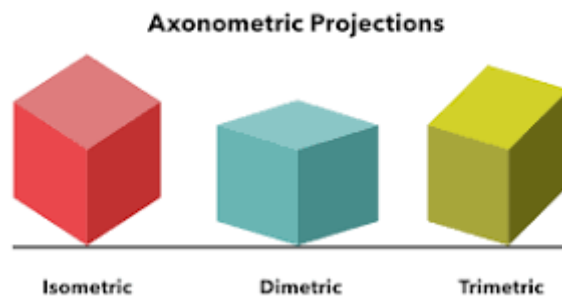


Figure 20: Axonometric Projections



Figure 21: Final Fantasy Tactics Advance, featuring a Dimetric projection

- Oblique projection, where all three axes are shown without foreshortening but has distorted diagonals and curved lines. Some games that feature this projection are SimCity200, Diablo and Bladur's Gate.

Oblique Projection

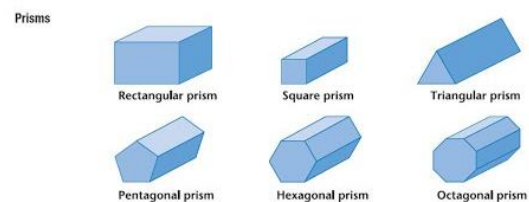


Figure 22: Oblique projection in different prisms



Figure 23: Simcity 2000 featuring a Oblique projection

- Billboarding is a technique that represents 2D images in a way that always face the camera line perpendicularly. This technique was used in systems whose hardware wasn't powerful enough to render fully 3D objects.
- Z axis scaling is a technique which scales sprites in relation to how distant are from the player, their size is diminished the farther they are and upscaled when closer. This creates a Z axis motion illusion. Sega OutRun is a game that features this technique by scaling the palm trees on the left and right side of the road so it creates a depth illusion where ones look closer to the player than others.



Figure 24: Sega OutRun

- Mode 7 was a system that allowed for a 3D effect featuring rotation and scaling while moving in any direction. It had not any real 3D models and was used on the SNES to simulate 3D graphics.

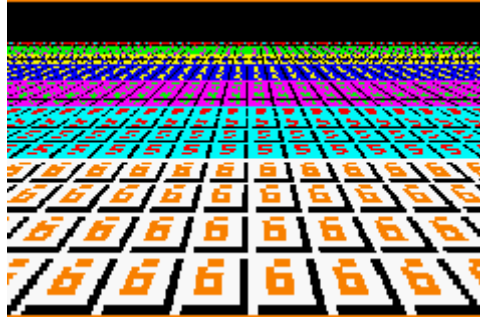


Figure 25: Mode 7 background featuring a texture to show the scaling to simulate depth

- Ray casting is a first person technique where a ray is sent for every vertical slice of the screen from the camera position. When a ray hits a wall or object, that part of the vertical screen slice gets rendered. The limitation of the movement of the camera and 2D movement field, this technique is considered 2.5D. It was sometimes implemented along with billboarding to show sprites facing the camera.



Figure 26: Ray casting rendered room from Wolfenstein 3D with billboard sprites and 2D field

Also, several chess games appeared using 2.5D graphics, one good example is Battle Chess, a very acclaimed and succesful game in which the Magic Chess project is inspired on. It is a chess game for computer where the chess pieces are represented by realistic figures which feature animations while moving or attacking. We can see an image of the game in the following figure:

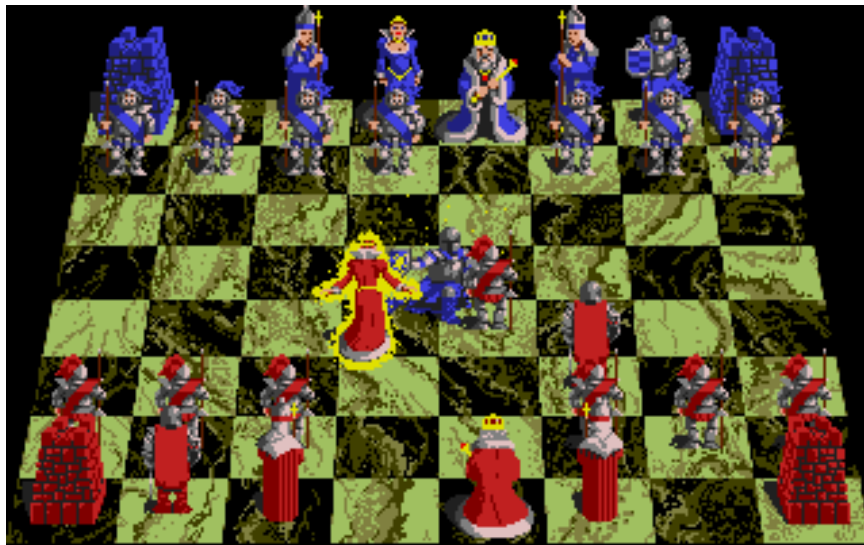


Figure 27: Battle Chess image

2.1.1.4. 3D Graphics

This kind of graphics use a three-dimensional representation of geometric data along with different techniques to enhance the look of it that we will enumerate in this chapter. This geometric representation will get transformed into a 2D image that will be presented to the display.

At the end of the 90s, the launch of the fifth generation of game consoles started pointing as real 3D graphics-based games as the way to go. Consoles with 32 and 64-bit architectures marked the start of the 3D Graphics era for video games. Further in this chapter we will discuss the basic techniques that were implemented in those consoles.

3D Graphics vary in the aesthetic they try to represent, and the techniques used to achieve the desired looks, we will focus on the evolution of 3D graphics regarding the creation of realistic images from 3D virtual data in real time.

The evolution from a hardware perspective focused on performance has taken huge steps towards being able to do several things, in simple words:

- Be able to represent more polygons in the screen to represent more geometric detail.
- Be able to load bigger amounts of data into memory such as larger textures or buffers filled with useful data for the shader programs to execute.
- Implement special architectures to perform certain operations faster.

We will take a look at some of the most important techniques in the following chapters.

2.1.1.4.1. Texture Mapping

The most basic technique used by the first 3D games was Texture Mapping. This technique allowed 3D models to have images as textures. It was possible through texture coordinates, which define how an image gets mapped to a geometric entity. Each Texture coordinate is associated with a vertex and indicate which texture point within the image should be mapped into that vertex.

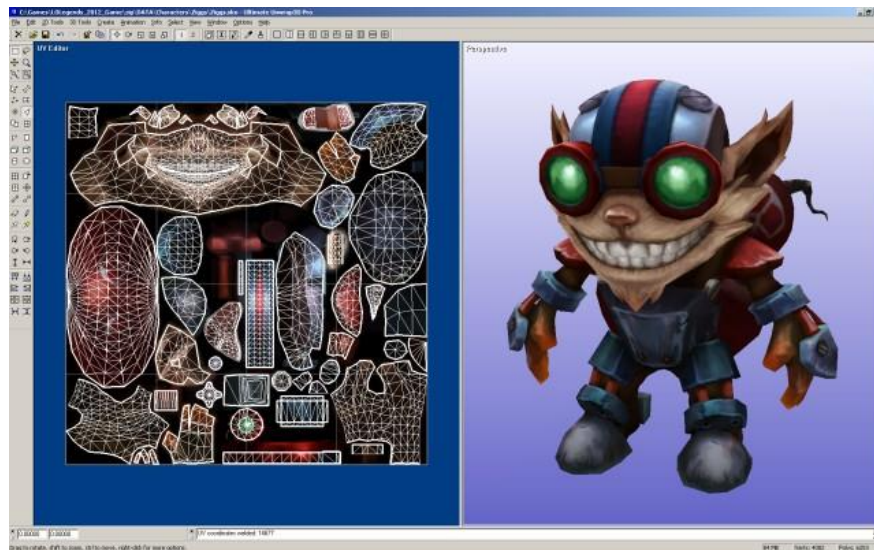


Figure 28: League of Legends character Texture Mapping. The left part of the image depicts the Unwrap of the 3D geometry over the texture

Texture filtering or smoothing was introduced, this technique samples the pixel's color by evaluating the value of nearby texels, which are nearby texture pixels. The filtering can be used for minification or magnification depending on the shape, size, angle, and scale of the object's visualization. Along the years different filtering methods were implemented such as Mipmapping, Nearest-neighbor interpolation, Nearest-neighbor with mipmapping, Linear mipmap filtering, bilinear filtering, trilinear filtering, and the more advanced anisotropic filtering.

A mipmap is an image that contains a sequence of the same image represented in different resolutions. It is used to reduce artifacts in rendering and increase performance, by choosing the correct image resolution to apply as texture to a model based on the depth of the 3D object.

Nearest-neighbor is the simplest filtering method. It uses the closest texel to the pixel center to set the pixel color.

Linear filtering samples from an individual mipmap while it interpolates the two closest mipmaps that are relevant to the particular sample. Bilinear filtering or blending takes the four texels that are nearest to the pixel center and sample them at the correct mipmap level. Then, the colors are combined by using a weighted average which is dependent on the distance. Trilinear does the same but combining the two closest mipmap levels using a linear interpolation after having done bilinear to each one of the mips.



Figure 29: Mipmap, sequence of the same image subdivided.

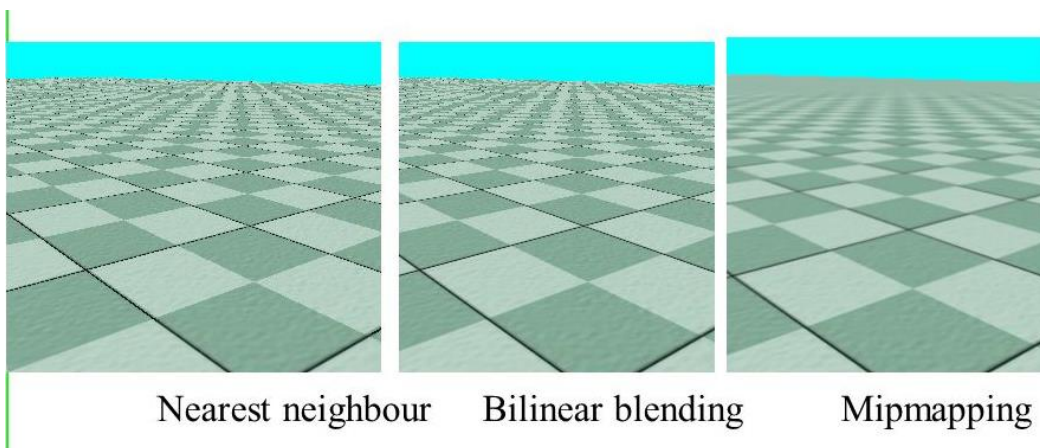


Figure 30: Nearest-neighbour vs Bilinear vs Mipmapping texture filtering

Anisotropic improves the quality of distant objects that are viewed at an angle by not using only square but samples the texture in a non-square shape which tries to map the footprint of the pixel.

Also, Texture mapping allowed for different techniques to be developed, such as Bump, Normal and Parallax mapping, which are techniques that applied to 2D textures in 3D rendering allow for extra geometric complexity simulation without the need for adding extra polygon density.

- Bump mapping is done by modifying the surface normals of an object and using a grayscale image plus the normals during the light calculation pass to simulate bumpiness in a surface.
- Normal mapping takes the normal vector and the light source direction towards the surface point and perform a dot product to get the intensity of light on that point.
- Finally, parallax mapping is a step further on the bump and normal mapping. It is a form of displacement mapping where the texture coordinates of the pixels are adjusted at render time to create a depth illusion that gets readjusted when the camera moves around the scene.



Figure 31: Bump, Normal and Parallax mapping in materials.

2.1.1.4.2. Lighting

Several kinds of lighting and shadows were implemented, from the most basic directional lights to the point, spot, and rectangular kinds of light. These light implementations allowed for the usage of different shading techniques such as:

- Flat shading, featuring a single color and normal per polygon face.
- Gouraud shading, featuring colors and normal per vertex, allowing for more smooth transitions throughout objects. Though this technique handles bad specular reflections because it might happen inside the polygon but not in the vertices, so the shading would not show it.
- Phong shading, which interpolates the color and normal per pixel, allowing for specular reflections.

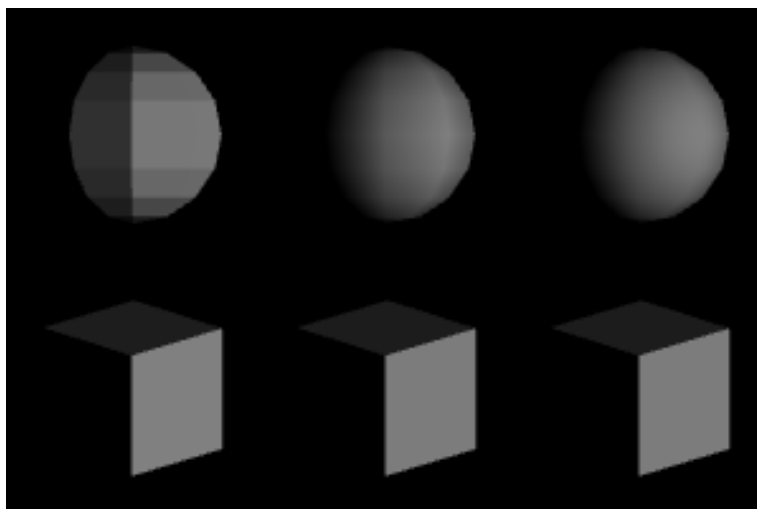


Figure 32: Flat, Gouraud and Phong shading models



And for different lighting models:

- Lambert Lighting Model
- Phong Lighting Model
- Ambient Light Model
- Blinn-Phong Lighting Model

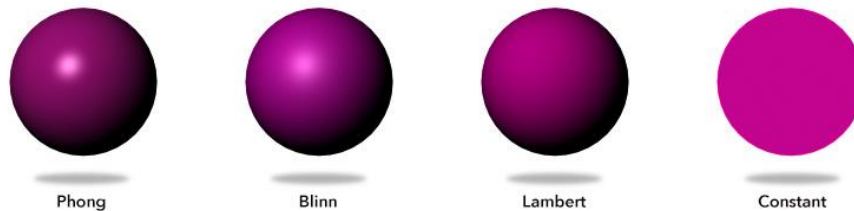


Figure 33: Phong, Blinn, Lambert and No lighting

Several Advanced Lighting models have been implemented, allowing for better look, featured in physically based rendering systems.

2.2. Recent History

In this chapter I will cover what I presume are the most important advances on videogame graphics in recent years, such as Physically-Based Rendering Materials and Real Time Raytracing.

2.2.1. Physically based Rendering Materials

A PBR material, which stands for Physics-Based Rendering, is a simulated material pipeline that can mimic almost any type of real surface to enhance a 3D model in particular. Numerous parameters are featured depending on the pipeline, including base color, metalness, and roughness. They can, for instance, mimic real materials like tiles, different kinds of wood, concrete, metal, facades, and backgrounds.

It accomplishes this by faithfully replicating refractions and diffuse, specular reflections between surfaces. It is an example of a technology that, as processing power increases globally, it is becoming more and more standardized. Although physics-based rendering techniques have been used in movies for decades, real-time 3D versions did not first emerge until 2007. Today, so many 3D artists, art professionals, and AAA games use this technique.



Figure 34: Example of PBR materials

PBR Materials present several benefits:

- Every parameter on the textured object is changeable, mostly through the use of UV-mapped textures. As a result, we may imitate a mixture of different material types with only one PBR material.
- Although often based on texture mapping, the structure allows for more complicated parameter definitions like procedural functions, making it flexible and scalable in terms of complexity.
- Getting 3D objects to look precisely how we want them to in all lighting situations may be challenging. PBR simplifies the process by specifying basic materials for various surface types and allowing the computer to determine how they will appear based on the lighting.

The micro-facet theory, on which PBR materials are based, describes a surface as a collection of small, perfectly reflecting facets whose orientation determines how "rough" the surface would appear at larger scales. The surface will seem diffuse if facet orientations are primarily random, whereas it will appear smooth and specular if facet orientations are generally aligned. The PBR material model's "roughness" parameter is its core and most significant one. Combining this with a good Fresnel effect simulation, which describes how much light is reflected given the view angle, both metallic and less reflective surfaces can be described properly with this model.

2.2.2. Real time Raytracing

Since the first articles on light simulations were published back in the late 1970s and 1980s, real-time ray-tracing has been a dream of graphics programmers. As of right now, this is a reality that is offered for sale in contemporary graphics cards (GPU) and gaming consoles like the PS5 and Xbox Series X.

Real-time ray-tracing is still a highly debated subject. There have been attempts to use denoising algorithms to reduce sample counts for rays, ranging from early papers and projects from the demo-scene using shader model 3 to perform basic ray tracing to more recent papers running into the problem of branch coherency during acceleration structure traversals, which are specially laid out in memory data structures that makes raytracing faster in certain GPU architectures, the most widely used is BVH which stands for Bounding Volume Hierarchy which depicts a tree structure containing a set of geometric objects.

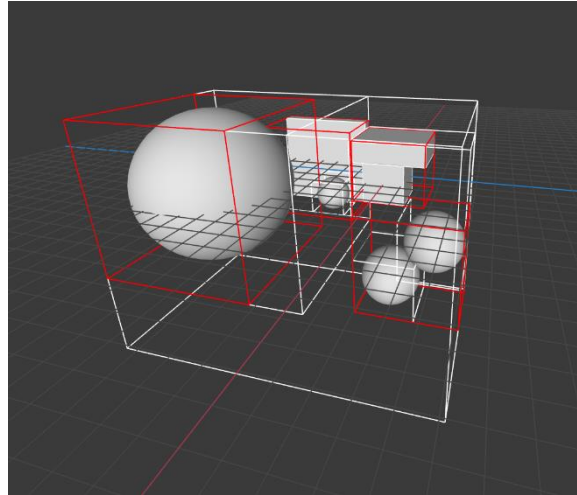


Figure 35: Bounding Volume Hierarchy visual representation

Although GPUs were initially designed to store frame buffers and perform basic rasterization and shading, they have since evolved into potent co-processors that can carry out non-specialized activities like computation, tensor, and ray-tracing procedures with variable degrees of hardware acceleration. Real time hardware accelerated ray tracing is now possible in modern high end graphics cards. Every raytracing-based algorithm follow the next steps:

1. Cast a ray based on a predetermined camera matrix for each pixel from which you want to extract radiance (light intensity) data. The world vector from the camera to the scene can be generated by taking the Normalized device coordinate of that pixel and multiplying it by the inverse view-projection matrix.
2. Use basic arithmetic algorithms to check for collisions between simple forms or use more sophisticated techniques like a bounding volume hierarchy to navigate the scene and a low-level acceleration structure as a KD-Tree to check each triangle of the scene's objects.
3. Test the surface's material function, also known as the BRDF (Bidirectional Reflectance/Transmission Distribution Function), to determine the radiation it emits.
4. By reflecting off that surface in accordance with the characteristics indicated by that materials or what you want to achieve: Global Illumination, Reflections, Shadows, etc.
5. After stopping, averaging, and writing at a predetermined quality level which determine the number of samples to use, the final color is obtained by averaging the radiance values of all the samples.



Figure 36: Realtime Raytracing featuring Star Wars troopers

2.3. Chess games graphics

I wanted to highlight several chess game graphics that I found extremely interesting in order to translate what they tried to represent into Magic Chess PBR Material system.

The first one is Battle Chess, which is a game released on 1988 by Interplay Productions in several systems: Amiga, Atari ST, Commodore 64, and NES. This game featured incredible animations at its time. It received a modern remake in 2015 called Game of Kings.



Figure 37: Battle Chess graphics.



Figure 38: Battle Chess remake graphics.

Chessmaster was a long running game series, they feature their own engine called The King engine, appart from displaying 3D graphics at its time (1995), the engine is notable for being able to implement different chess playstyles.



Figure 39: Chessmaster 3D graphics

Battle Vs. Chess was released in 2011 and featured a high fantasy theme along with 3D graphics and different scenarios that acted as chessboards. Each piece had their own attack, from hammer hits to magic spells.



Figure 40: Battle vs Chess graphics

Finally, Chess Ultra was released in 2017 for PS4, PC, Xbox One, HTC Vive, Oculus and Nintendo Switch. The game supports 4K resolution and has different board designs along with rooms where the chess games happen. It is thought to be the best way to play traditional chess in video games



nowadays. It also features multiple AI levels, online play and several puzzles such as revisiting classical games from a grandmaster perspective.



Figure 41: Chess Ultra graphics

2.4. Raytracing Material Library

The Raytracing Material Library for Magic Chess is a project focused on developing materials that take advantage of actual technologies such as Raytracing, or Unreal Engine 5 capabilities as Lumen or the material pipeline the engine has. As we will see in the Development section of this document, I have implemented several materials and effects such as different water materials, glass, or plastic variations which uses the different Material systems within the engine.

Unreal Engine 5 is very customizable engine which allows developers to adjust it to the needs of the project. Thinking on that, I also started the creation of different sublevels, each containing different configurations for raytracing and various effects.

Even though Raytracing now is very powerful, it is still a relatively new technology regarding real time rendering systems, it needs very powerful hardware to run smoothly with an austere configuration.

3. Objectives

The idea behind the Magic Chess project was already defined when I started the development of the Material Library for the game. My role was to support Raytracing and several effects derived from it in the materials that can be used within the project.

My objectives can be divided into several key points decided at the time of my proposal:

- Get to know the different effects such as reflections and refractions, as well as the state of the art in Realtime Raytracing as well as Game Graphics.
- Create a robust architecture within the materials that allow for further iteration during the game development process.
- Create a series of basic materials that use Raytracing such as glass or plastic.
- Implement effects to those materials to create a visually appealing set of materials such as frosting, reflections, caustics...

During the development of the Magic Chess project, my main objective was to create a material system which enhanced the visual aspect of the game, making use of raytracing and other cutting-edge technologies. This main objective can be broken down into the previously stated four points.

4. Methodologies

Eclipse Games team is composed of two programmers, two artists and two designers as well as another student who is working on the artificial intelligence for the same project, I have been developing the materials. Other people are in charge of sound and audio development, but I haven't meet them as it was not needed for my workflow. I have assisted to some of the meetings where the mood and aesthetic for this project were decided and iterated upon, as well as giving feedback regarding those decisions.

As for the working methodology, I followed an agile approach where fast two-week iterations were done, with the help of tools such as HackNPlan, as we will see in the next chapter, which helped me divide my work into smaller subsets of tasks that I could complete in my sprints. In Eclipse Games, there is a scrum meeting each morning to discuss what was done the previous day and have a general sense on how the project is going, as well as being able to provide other team members with suggestions or help if needed.

4.1. Tools

The contents of this chapter are the different tools I used to develop this project and to work with Eclipse Games.

In order to communicate we used Discord, which is a reliable messaging tool which also allow for private servers and voice chats where you can share your screen or view other's. It was also chosen due to the lack of a physical office, and different members of the team living in different cities.

At first, we started the research process using Unity's compute shaders, as I wanted to explore more about how data structures could be laid out in order to speed up the raytracing by software.

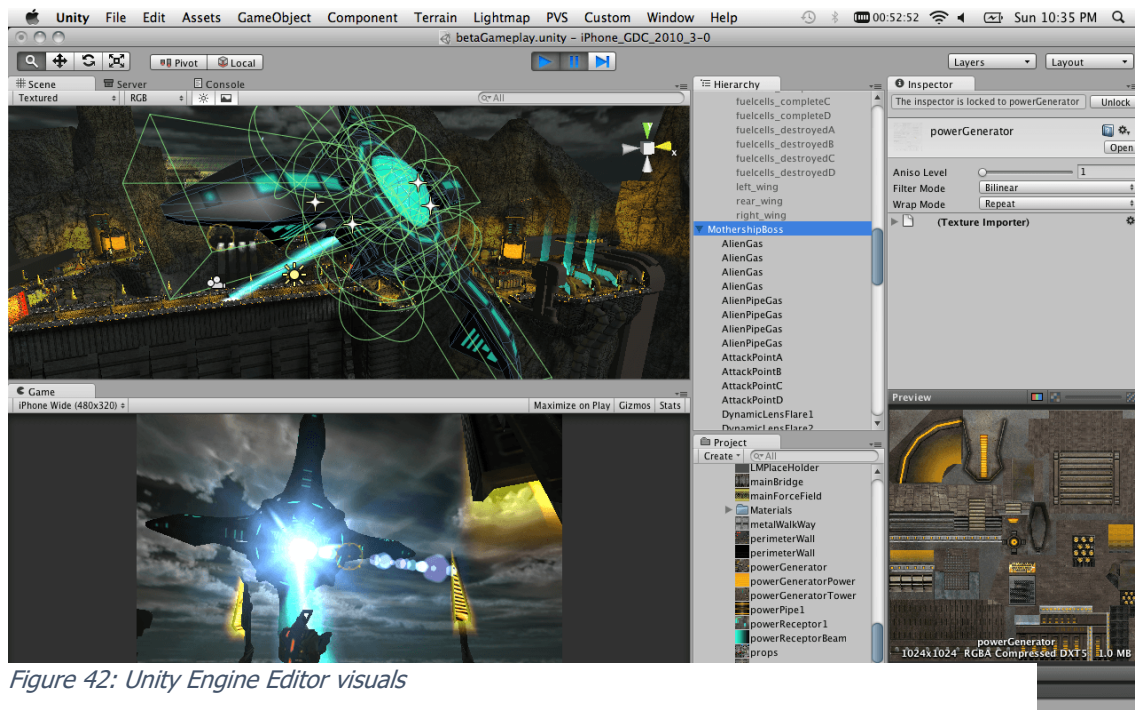


Figure 42: Unity Engine Editor visuals

This turned out to not be powerful enough to support real time as we will see in the development chapter. Though it served me as a foundation regarding Acceleration Structures such as the Bounding Volume Hierarchy Acceleration Structure discussed in Nvidia developers' site as a way to speed up hardware raytracing even more.

Unity is the game engine where Eclipse Games has developed the majority of its games so it was common sense to start from there, even if later we would change the development engine, as we did. Unity is a generalist engine which has on its back a huge number of successful titles such as Hollow Knight, Cult of the Lamb, Several Pokémon games, Legends of Runeterra... As we can observe, Unity is the base of games ranging from Metroid Vania genre games to card games.

Unity is also not open source, this means that as a developer, you can't see how the internal implementation of systems work, which would serve a lot in our use case, as Raytracing is a very performance exhaustive technique.

Unreal Engine 4, on the other hand, was open source and starting to focus on its raytracing capabilities, that, added on top of its out of the box visual and lighting quality made us change the development engine quite early in development. This was not an issue to me as I had worked in that engine in the past and was familiar with it. Along the way, Unreal Engine 5 was announced, extending the raytracing capabilities among other systems. This made us update our project to use it, as it was just upgrading the engine and we didn't have many systems laid out, it turned out to be quite easy due to the architectural design of Unreal Engine.

Unreal Engine is a game engine capable of producing AAA quality games if used correctly, and quite nice games if used in a user level. A plus on its design and philosophy is that it is open source, so you can delve into the code to search for examples or change and recompile the engine, which is very costly and requires of a powerful machine. This engine featured games such as A Way Out, Astroneer, Back 4 Blood and Gears 6 among other titles.

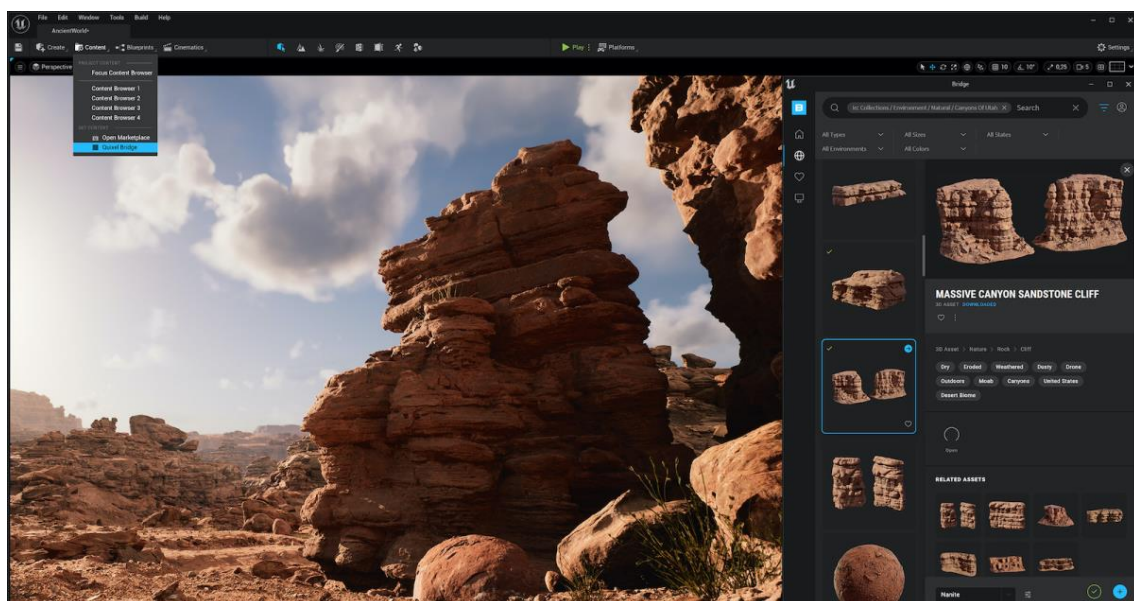


Figure 43: Unreal Engine 5 Editor Visuals

Unreal Engine is developed in C++ and its development language is C++, so I decided to use Visual Studio 2022 to work with it. However, as my tasks were mainly developing a material

library which used Raytracing, I didn't have to delve much into C++ apart from some quick fixes and small changes for the code to work properly. VS2022 IDE switched to a 64-bit application architecture which made it work significantly faster than Visual Studio 2019, so the development within that IDE was fantastic.

As for the version control software, I used what Eclipse Games has been using for quite some time, SourceTree, which is a visual interface for tools like git or mercurial. I also delved on my own in software that used a different framework such as Perforce, which works particularly well with unreal engine due to the exclusive checkout of binary files, so difficult merging bugs get avoided.

As a Ticket manager system, I chosen HackNPlan, which was introduced to me by Eduardo Jiménez, I found the tool really useful because it allowed me to create different boards and tickets depending on the tasks I needed to do. It also allowed us to introduce time information as well as priorities to structure the developing times.

It also allowed me to know which areas of the development needed more work and time into them.

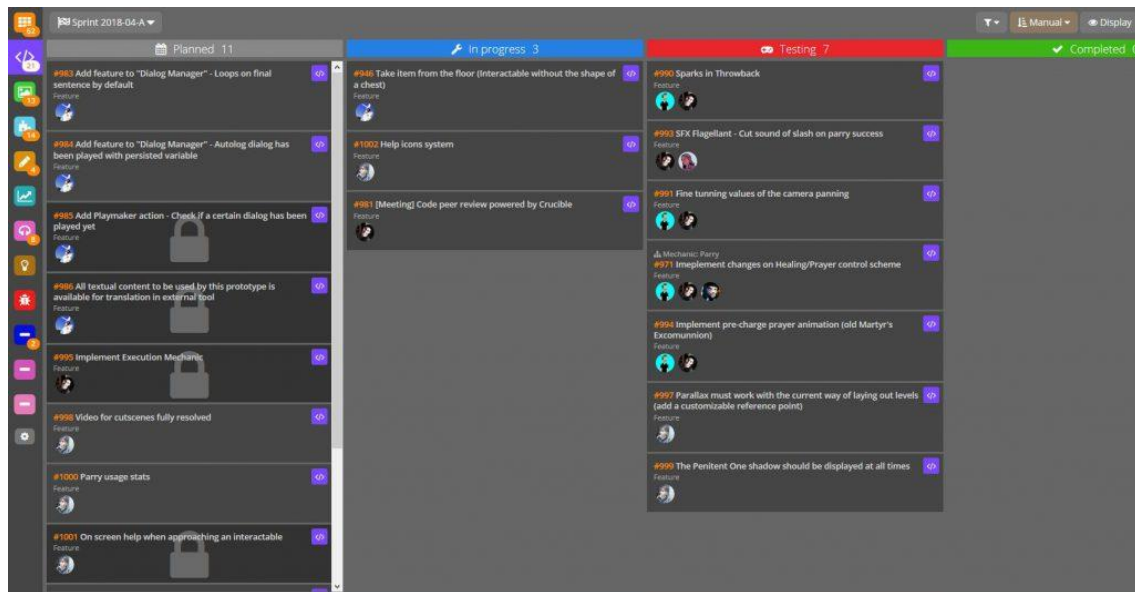


Figure 44: HackNPlan Interface

5. Development

The following chapters illustrate how the different materials and effects from the Material library were implemented and how Unreal Engine 5 handles Raytracing so the material pipeline can make use of it. Also, the first chapter is dedicated to the prior work done in Unity.

Videogame development follows an iterative process. Materials follow the same approach, as assets intended to be used in games, at least for our use case, they need to change constantly to comply with the game needs. These needs can be adjusting some values to get a slightly better result, or even packing textures in a way that the material rendering doesn't take an excessive amount of budget.

5.1. Raytracing Research and Development in Unity

Previous to the work within Unreal Engine 5, and as a research job, I implemented a prototype Raytracer in Unity using compute shaders.

Compute shaders are general purpose programs that can be executed in the GPU to make parallel calculations. This allowed me to speed up the raytracer a little, however it was far from being performant on this state, as it was not using any acceleration structure and at that time, the GPU I owned was not capable of hardware raytracing, so it was a software Raytracer.

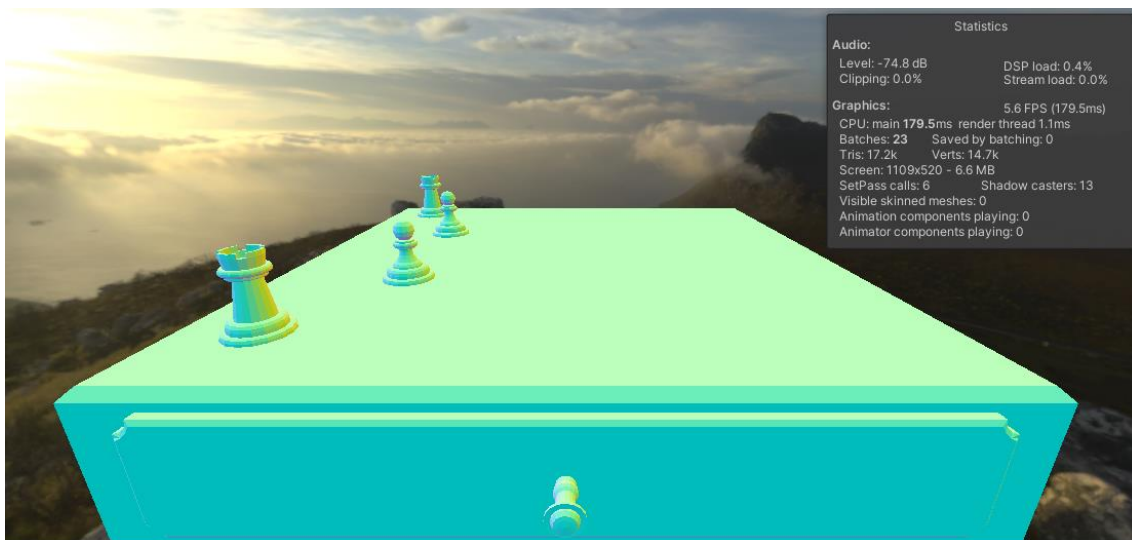


Figure 45: Compute Shader Raytracer in Unity.

As we can see in the previous image, it was not performant, the full viewport resolution was running at six frames per second. However, lowering the resolution would increase the framerate significantly, but not as much as it was needed to be usable.

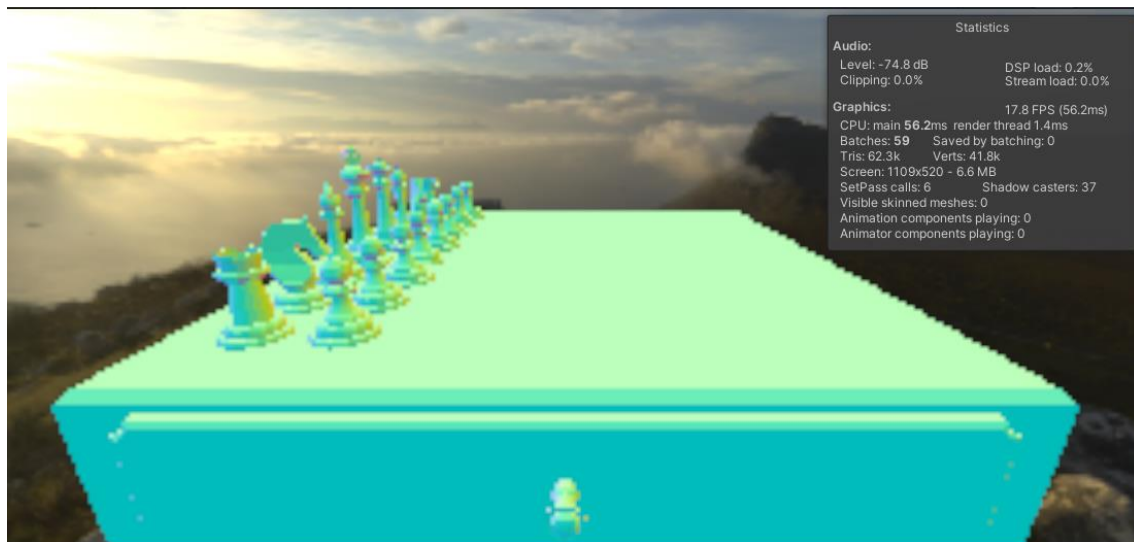


Figure 46: Lower Resolution raytracing.

In order to speed up the raytracing algorithm I researched about acceleration structures and algorithms. The principle of these structures is to help deciding quickly which objects that are present on the scene are likely to intersect with the ray and reject a large group of objects which we know the ray will never hit instead of checking the ray against every triangle.

One of the most used acceleration structures in different papers related to raytracing and software raytracers are KD-Trees (K-Dimensional Trees).

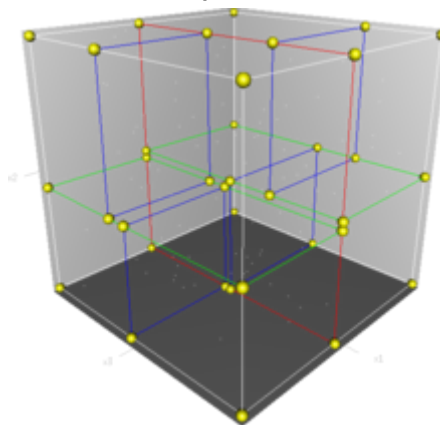


Figure 47: 3Dimensional KD Tree.

This particular tree is a space partitioning data structure to organize points in a k-dimensional space, in our case it would be 3 dimensional. As we know, 3D meshes are composed of triangles, and triangles of points. The 3-dimensional tree divides the scene with planes parallel to the scene cubic bounds by considering the areas that are more triangle-convoluted. The traversal algorithm for this tree takes as input a tree and a ray, then searches for the first primitive that is in the tree and is being intersected by the ray. The traversal starts at the root node, and usually a stack is used to keep track of the nodes that are left to visit ordered by priority.

These techniques are commonly used at offline rendering, for example in the film industry, to generate very complex frames without having to wait that much time for the samples to be gathered. Even though it is faster, it is not suitable for real-time rendering as it is a software technique.

However, there are raytracing capable GPUs, which have built in operations to squeeze the full potential of the acceleration structures. Nvidia implemented a Bounding Volume Hierarchy in their RTX series to speed up real-time raytracing. The BVH is a tree structure that contains geometry,



like the KD Tree, but with the difference that each leaf contains a bounding volume that contains objects. Then, these tree nodes are grouped into small sets and then stored in larger sets, which are grouped recursively and included into larger bounding volumes, creating a structure that has only one bounding volume at the top of the tree.

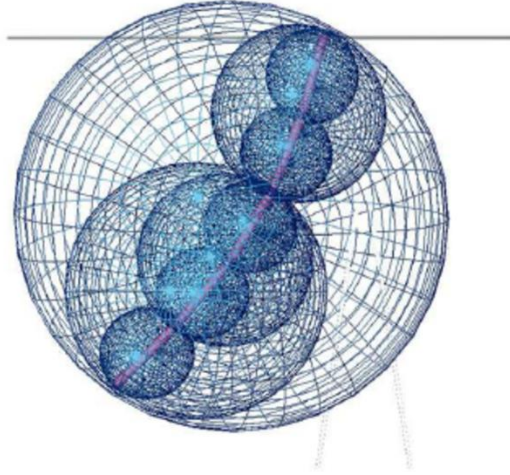


Figure 48: Bounding Volume Hierarchy

After some time researching on Raytracing technology and material systems, we decided to switch the development to Unreal Engine 5, as we will see in the following chapter.

5.2. Available Technology, why Unreal Engine 5.

Very early in the development of the Magic Chess project, we decided to migrate over to Unreal Engine 5. This happened because Eduardo Jiménez, Daniel Gracia, a teammate, and me agreed on it in order to test the new technologies that Epic Games was releasing in their engine and facilitate the development.

Unreal Engine's lighting quality out of the box is better than Unity, the engine we were using previously, so that was a plus for me. I had done previous work within Unreal Engine, so it was not completely new for me. However, many of the systems that are present in Unreal Engine 5 do not work the same as their counterparts in Unreal Engine 4, so further research on them was needed.

The UE5's material pipeline that we will see in the following chapters is highly customizable and allow from simplistic to really complex materials. From material instances derived from a father material in order to change the base color to a layering system where different material layers get blended using user-defined blending functions.



Figure 49: Pawn from Magic Chess with simple material using texture maps.

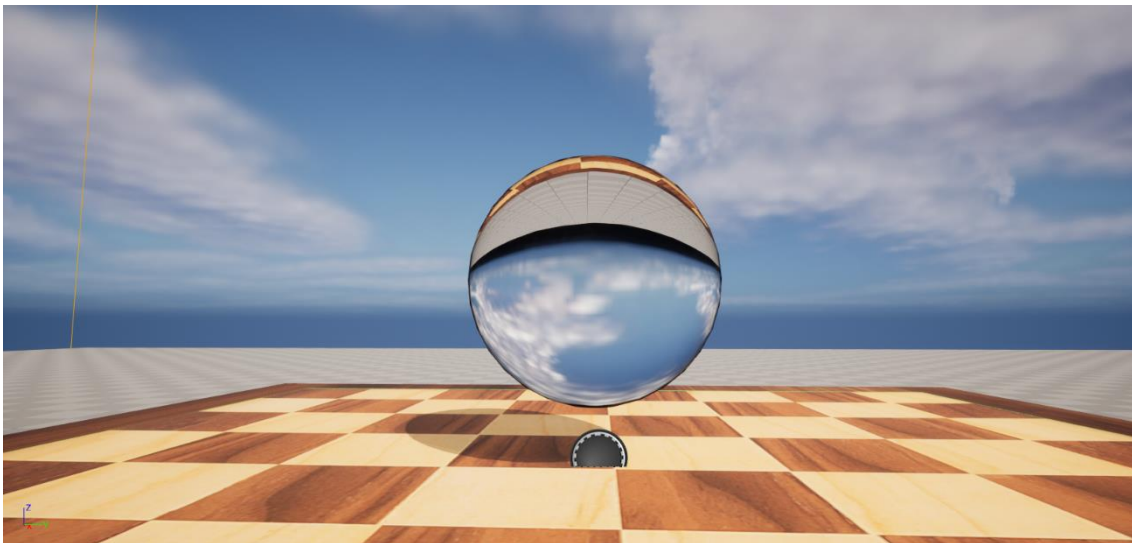


Figure 50: Glass Material featuring Raytraced Reflection and Refraction

5.3. Master Material and Material Instances

The Master Material and Material Instances concept is a hierarchical architecture where a Master asset (The Master Material) has the base configuration that all materials of a type of share.

This is not always possible to do, but it is usually very helpful because the changes on the master material propagates through all the material instances that are child of that master. It is also inefficient to create a completely unique Material for each Actor in a project, especially given that related assets frequently need Materials that are also similar, in our particular case, we follow a coherent aesthetic, so chess pieces materials will be similar from one another if not all derived from the same parent.

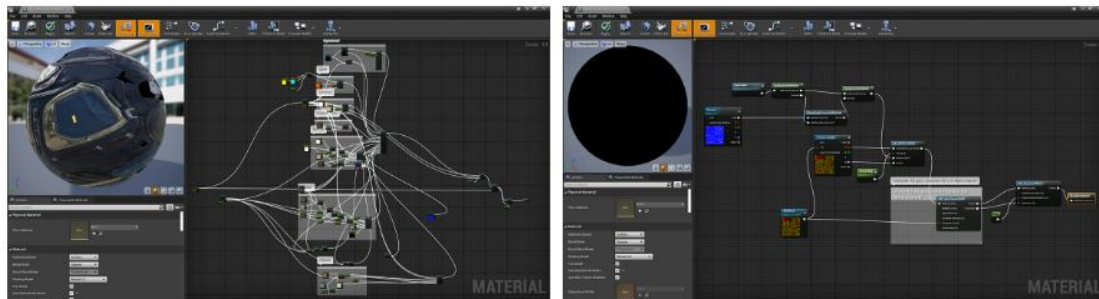
Materials are shaders that instead of being written in HLSL directly, they are composed via the visual scripting tool given by UE5. However, if we wanted to write custom HLSL code we could still do it by using a Custom node, though it is hardly necessary due to the high amount of functionality that exists currently in the engine. Each node in the visual scripting tool is referred to as a Material Expression, which is being translated into HLSL under the hood.

Similar to visual studio's disassembly capabilities, we can access the HLSL generated code through Window > Shader Code > HLSL Code.

5.4. Material Layers and Material Layer blends

When creating materials for the Magic Chess project, the UE Material System provided a strong and scalable solution. The technology enabled me to layer and combine several textures to provide a special material for items placed in the levels. By combining and blending textures using Material Layer asset types in the material graph. By using Material Instancing, this method streamlines the creation of layered and blended materials.

Using Material Layers allows us to design materials that might otherwise be too complex while keeping them far more manageable in terms of future editability. Although the same sort of material could be created using the normal Material process (without layers and Material Functions), the Material Layers workflow offers superior control, flexibility, and less complexity in the material creation process.



Base Material using Material Function workflow

Base Material using Material Layers workflow

Figure 51: Image from Epic Games showing the different workflows.

5.5. Material Functions

Material Functions are little material graph fragments that may be packaged and utilized again across different materials.

The development of materials is made simpler by the use of Material Functions, which enable complicated Material Graph structures to be saved out and rapidly reused in other Materials as well as the abstraction of more complex networks into single nodes.

In the Magic Chess project, the Material Functions have been used to decouple logic that was being used in several materials like the water, where a custom material function to control the influence of each of the normal textures for the final combined result was used.



Figure 52: Water material

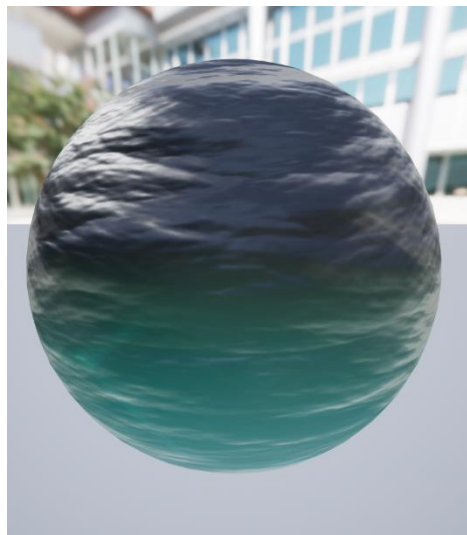


Figure 53: Murky water material

5.6. Plastic materials

Plastic is a material that can be made from a very wide range of organic polymers, and molded into different shapes, kept soft or rigid..., from a graphics point of view, plastic is a surface that can range from being very glossy to being completely rough, translucent, reflective...

In order to implement that kind of surface, I created a Material Layer for the plastic base material and set some default parameters along with some adjustments which led to the following results that can be seen in the next figure.



Figure 54: Plastic-Like materials that feature frosting, glossiness, and translucency

I managed to generate four types of base plastics:

- A very rough one, which is the rightmost and looks soft, it was achieved by setting a higher roughness value.
- A glossy one, which is the yellowish, where the roughness value is much lower.
- One semi translucent, which was achieved by reducing the opacity of the object, lowering the roughness, and adjusting the index of refraction to match real life materials that were alike.
- The last one looks similar to hard candy, it is similar to the semi translucent, but with a slight lower roughness value and a little bump in the metallic.



Figure 55: Plastic materials with a mirror surface underneath



Figure 56: Plastic materials view from a different angle.

As raytraced reflections need to bounce from one object to another, sometimes the reflections get artifacts due to the number of bounces not being enough to catch the object that is there. But it is something expected. Increasing ray bounces rapidly hinders the performance.

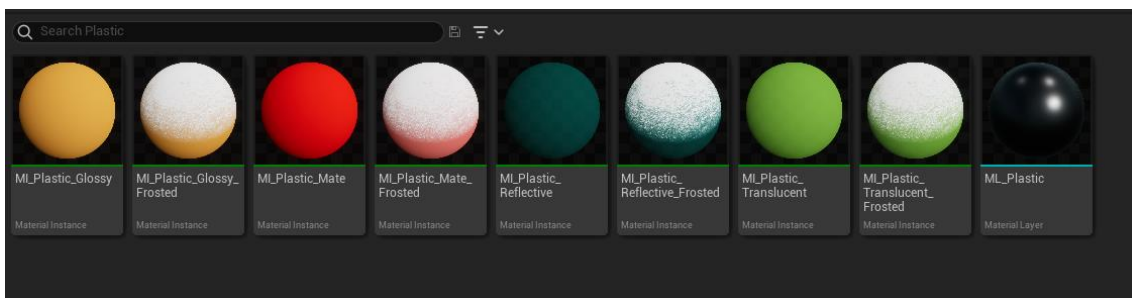


Figure 57: Plastic Material Instances



Figure 58: Pawn chess model from Magic Chess with different materials.

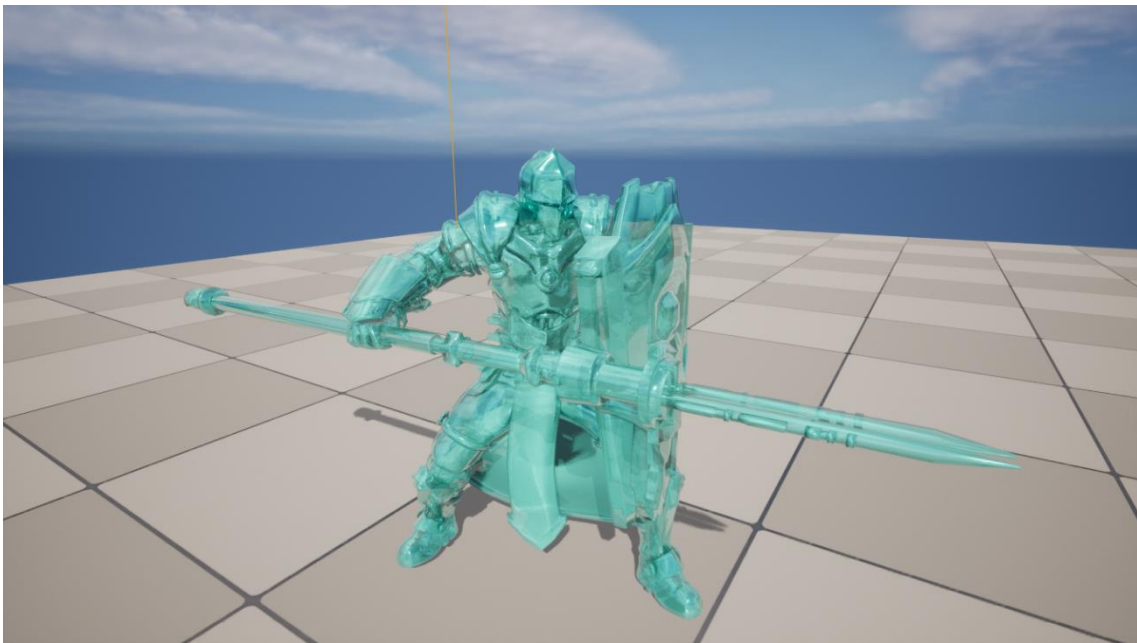


Figure 59: Pawn with colored translucent glossy plastic material.

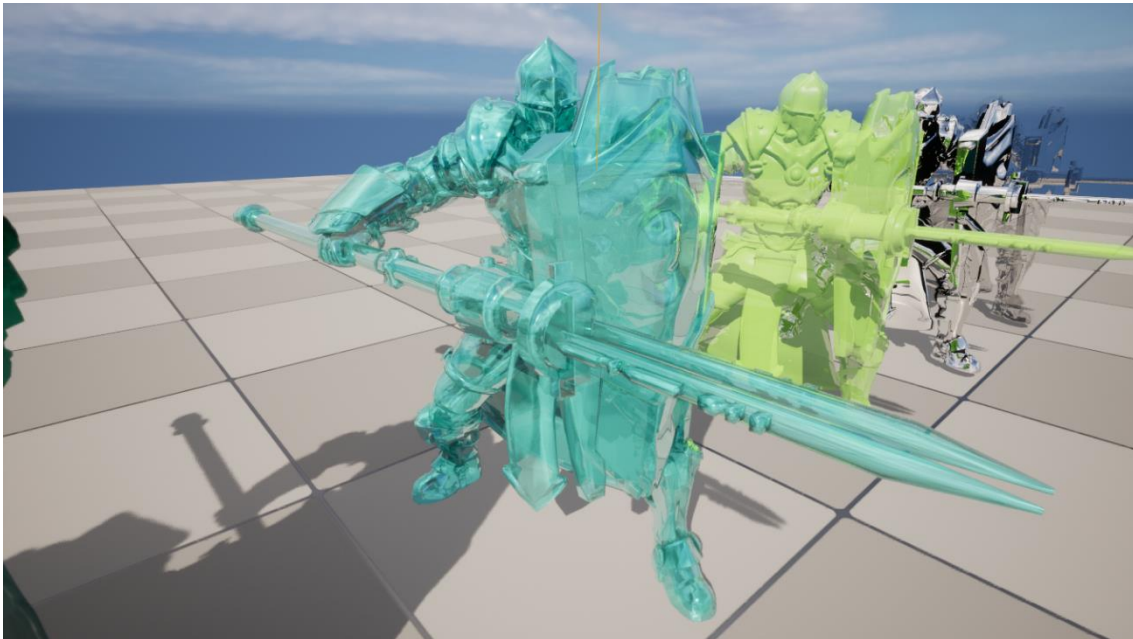


Figure 60: Several pawns from MagicChess project.

5.7. Glass materials

Glass is a non-crystalline, frequently transparent amorphous material that is widely used in windowpanes, dinnerware, and optics, among other practical, technical, and decorative applications. Glass is most frequently created by rapidly cooling the hot form; other glasses, such as volcanic glass, are created spontaneously.

For this chapter we will focus on clear glass, which is the material that I have implemented completely and is giving better visual results, in the following figure we can see a full chess set featuring the clear glass material.

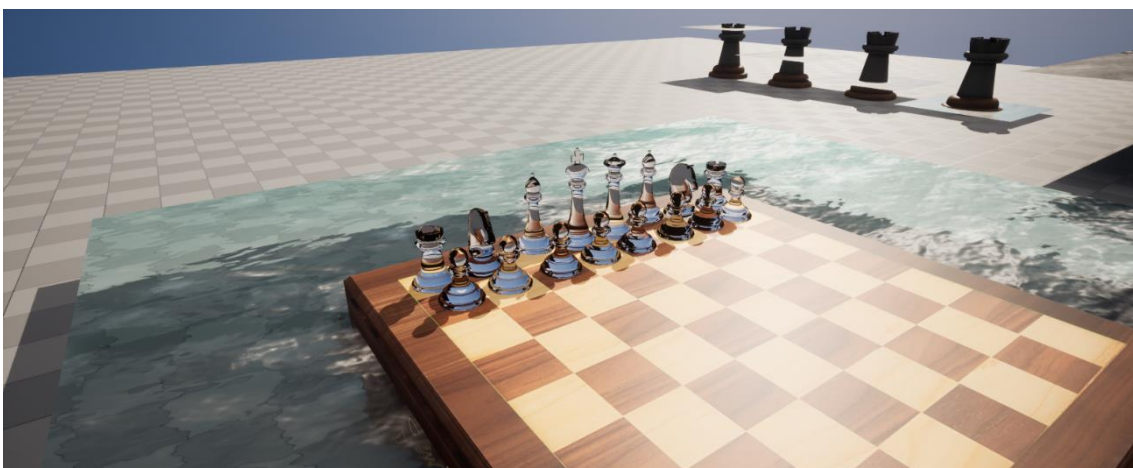


Figure 61: Glass material applied to a chess set

In order to implement glass, I started looking for the physical properties of it and I found several refractive indexes, densities, ratios..., and started testing all of them until I found what suited best with the volume simulation. The best refraction index turned out to be around 1.55.



However, with different material values other indexes could work, as there are different types of glass. For instance, with 1.55 as a refraction index, the clear glass works well, but the stained glass can't be properly simulated, so the hard candy plastic turned out to give better results in that field.

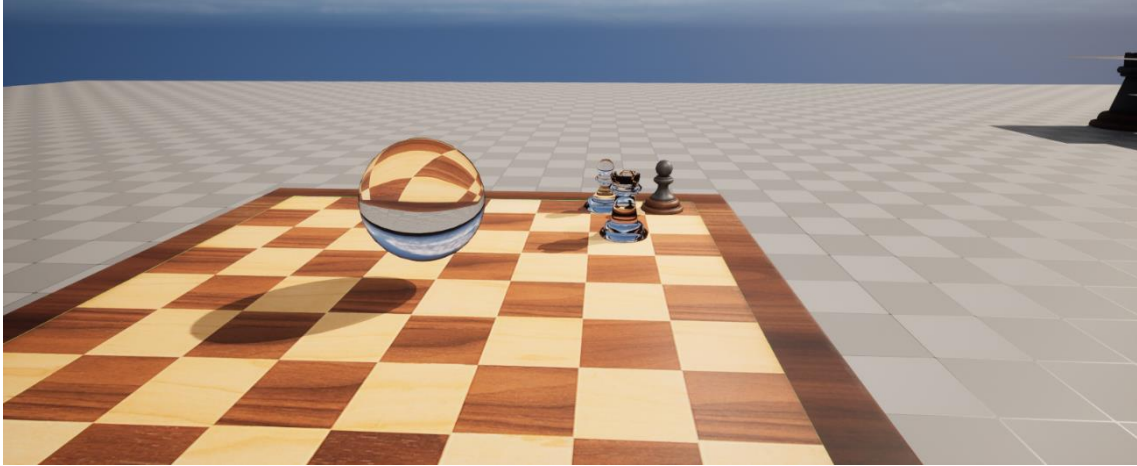


Figure 62: Glass sphere

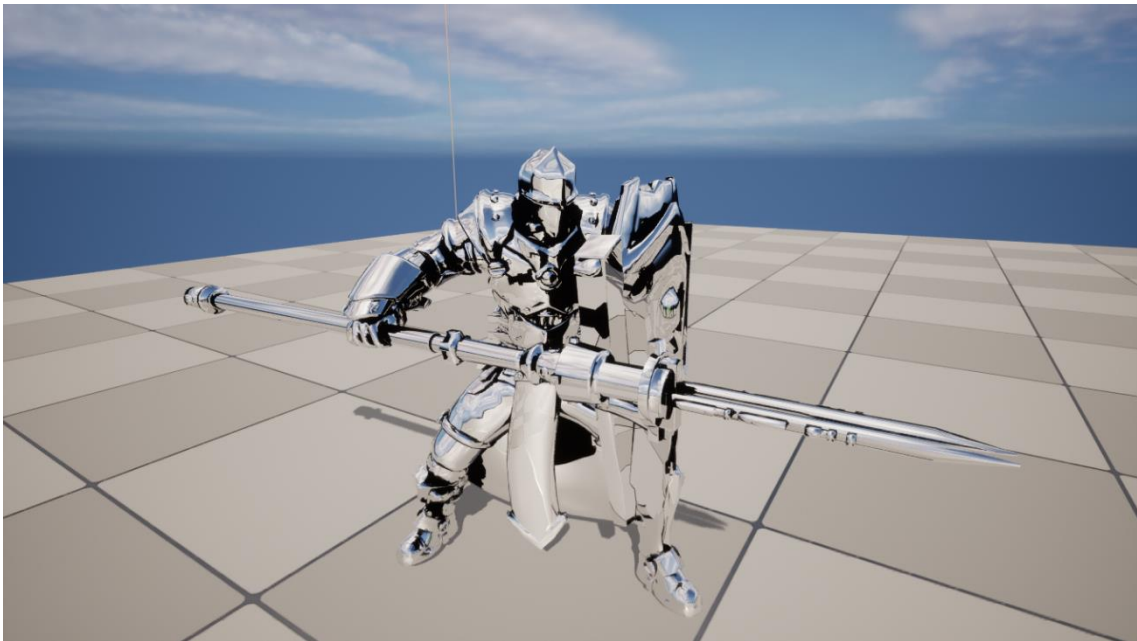


Figure 63: Pawn model with mirror material applied.

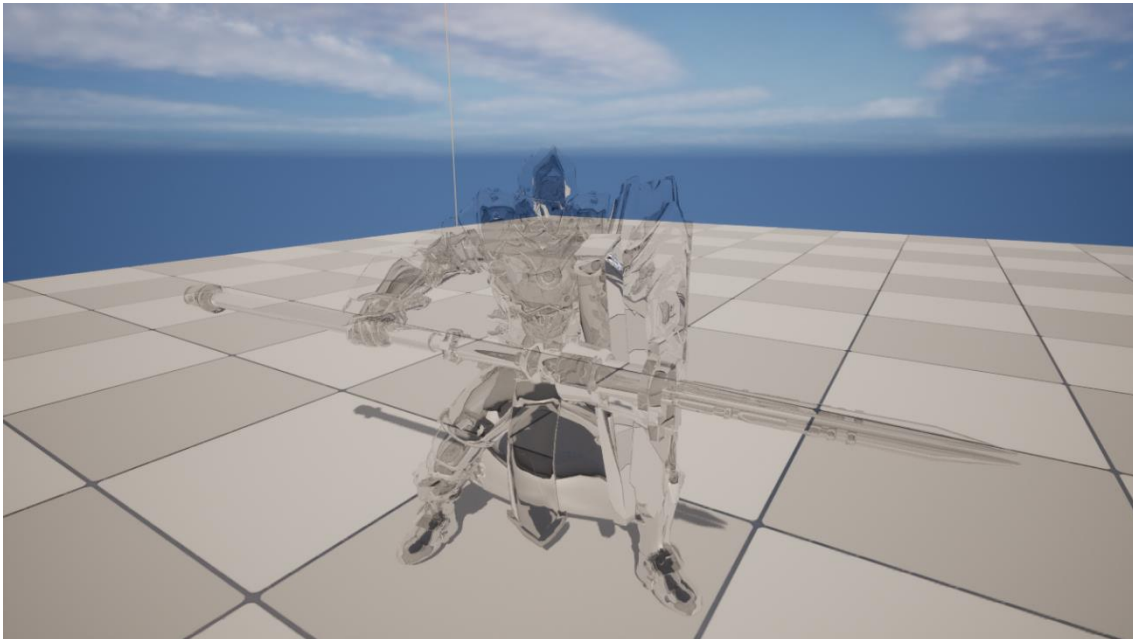


Figure 64: Pawn model with clear glass material.

5.8. Water materials

Water is a somewhat transparent liquid that can be similar to a glass if it is still and unperturbed.

In order to simulate the water material, I used some techniques:

- Normal texture combination and weights, where two or more normal textures are loaded and given a certain weight and offset to it each frame, and combined, which gives the illusion of the water moving.
- Water ripple simulation with close objects by adjusting the refraction index to a sine wavelength when an object is closer.
- Water caustics simulation by animating a contrasted noise texture each frame.
- Water Scattering and Absorption of color wavelengths to allow for underwater foggiess by using UE5 Water material output values.

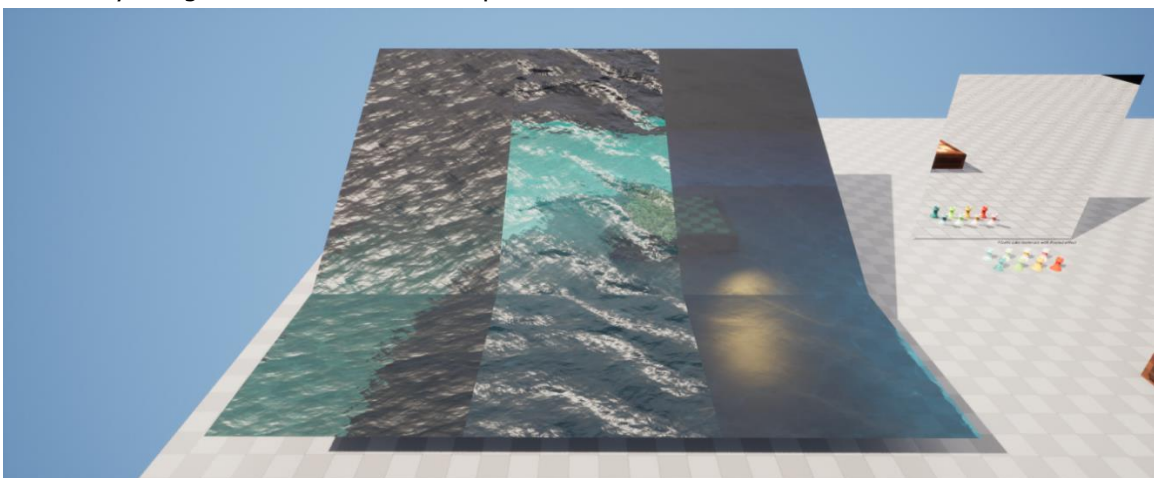


Figure 65: Three water material variations next to each other.

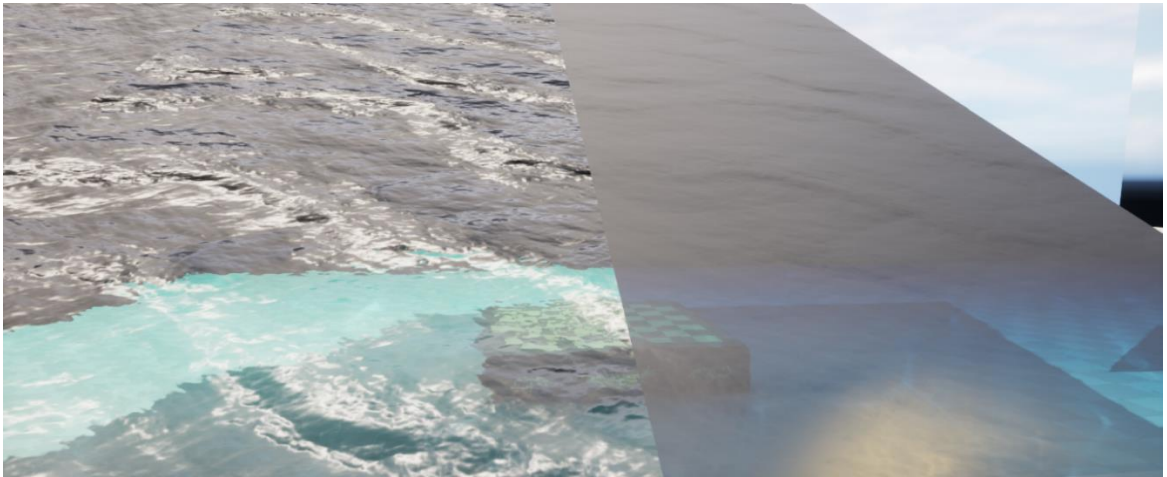


Figure 66: Water materials featuring murkiness.

I implemented three variants to the water material by using the material instances:

- Clear water, where the fog is adjusted to not be so intense and the absorption wavelength is more permissive with colors.
- Murky water, which is more obscured and has a little opacity to tune up the darkness, simulating turbulence in the water.
- Unperturbed water, which is a variation with very soft normal participation, simulating a calm water environment with almost no disturbance.



Figure 67: Water material featuring normal combination and animation, ripples next to geometry and refraction.

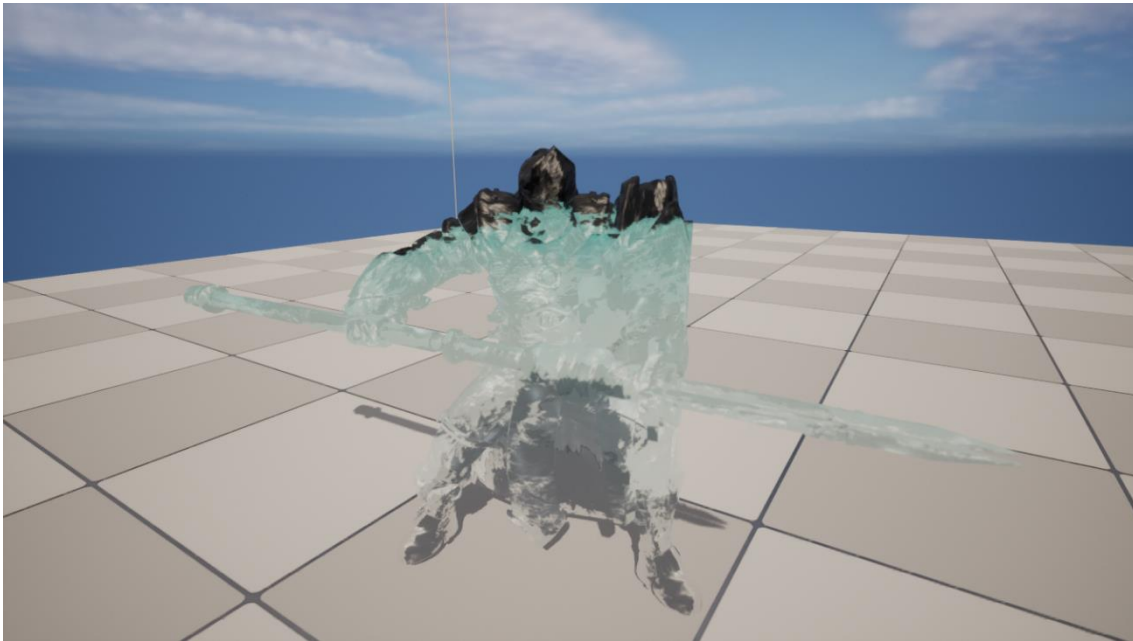


Figure 68: Pawn model with clear water material.



Figure 69: Pawn model with murky water material.

5.9. Dust/Frost

This effect stacks as a layer on top of other layer materials, generating visuals as if some dust or frost had gathered in the upper part of the surface.

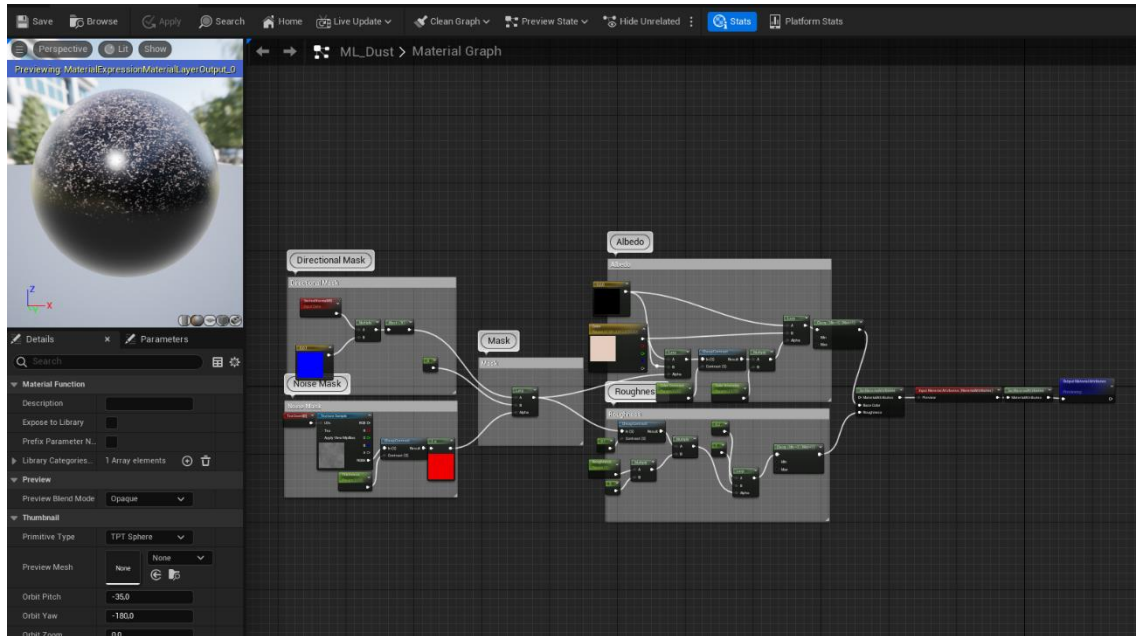


Figure 70: Dust/Frost Material Layer effect, featuring a directional mask.

As we can see in the previous figure, to create this effect, I first calculated a directional mask that will map a noise texture to the top of the surface the material gets layered on. After that, I created a color parameter to be able to distinguish between bluish white for frost and other colors for other effects such as dust. Also, depending on the value of the mask, the roughness gets readjusted to be more mate when needed because it would not make sense for a glossy surface to have glossy dust. The dust is rough.

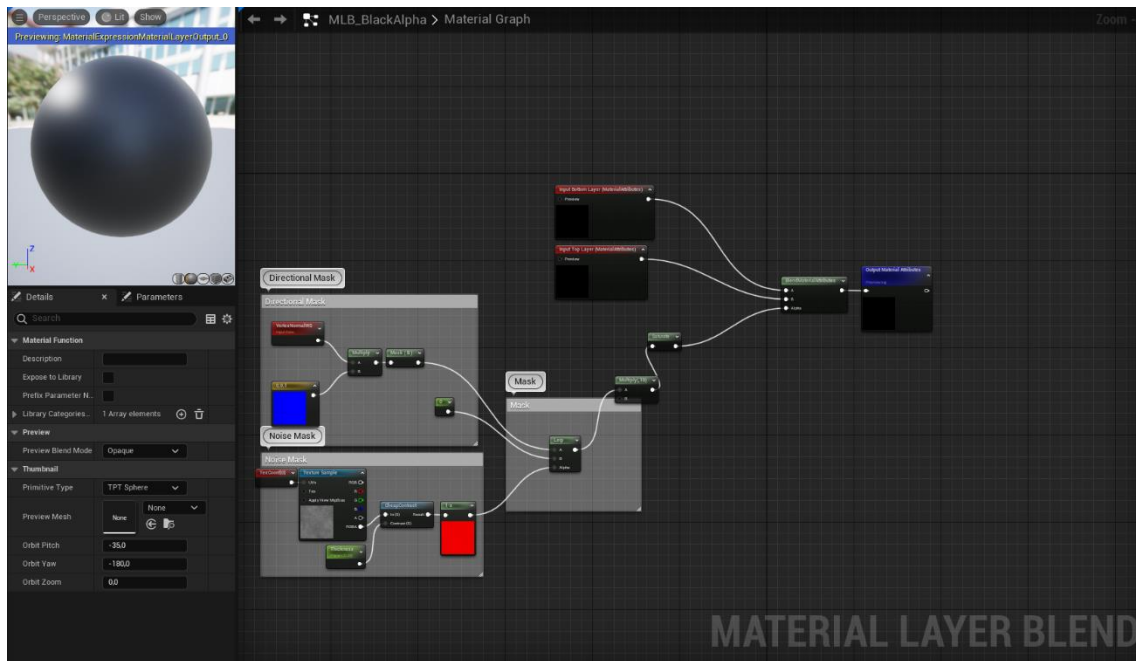


Figure 71: Material Layer Blend for the Dust/Frost layer



Figure 72: Pawn with colored translucent glossy plastic material with frost.



Figure 73: Pawn with glossy plastic material with frost.

5.10. Post-process volume sublevels

By combining a variety of attributes and features that impact lighting, tone mapping, coloring, and other aspects of the image, post-processing effects provide artists and designers the ability to create the overall look and feel of the picture. To use these functionalities, a Level may be introduced with a particular kind of volume referred to as a Post Process Volume. To define the appearance of a particular region, several volumes can be used, or the volume can be configured to affect the entire picture or even overlap volumes to generate transitions between areas.

Unreal Engine 5 allow for asynchronous loading of sub levels, so different configurations for raytraced effects can be set inside the same level without the need for reloading it or restarting the game project.

For Magic Chess this means that we can achieve a better look without losing too much performance if we use this system to keep the frame time in budget by selecting which effects we want to execute together and which ones we don't want to execute at all as Raytraced effects are performance exhaustive and even optimized by hardware can be slow sometimes.

The Post-process volume features among others:

- Depth of Field, which blurs a view based on the distance in front of or behind a focus point, similar to what happens with real-world cameras. Based on depth, the effect is employed to direct the viewer's attention to a particular topic in the frame. Additionally, it gives the produced image a style that gives it a more photorealistic or cinematic appearance.
- By simulating the glow surrounding lights and shiny surfaces, bloom further increases the apparent realism of the produced image. Lens flares and dirt masks are two additional effects that work with bloom but are not in the UE5 bloom features.
- Screen Space Effects such as SSReflections or SSAmbientOcclusion.
- Raytraced effects, that will soon get replaced by their Lumen implementations, such as RTReflections, RTAmbientOcclusion, RTTranslucency and RTGlobalIllumination.
- Lumen is the default global lighting and reflections system in Unreal Engine 5, a completely dynamic global illumination and reflections system created for next-generation consoles. In huge, intricate settings, Lumen displays diffuse interreflection with infinite bounces and indirect specular reflections at many sizes.

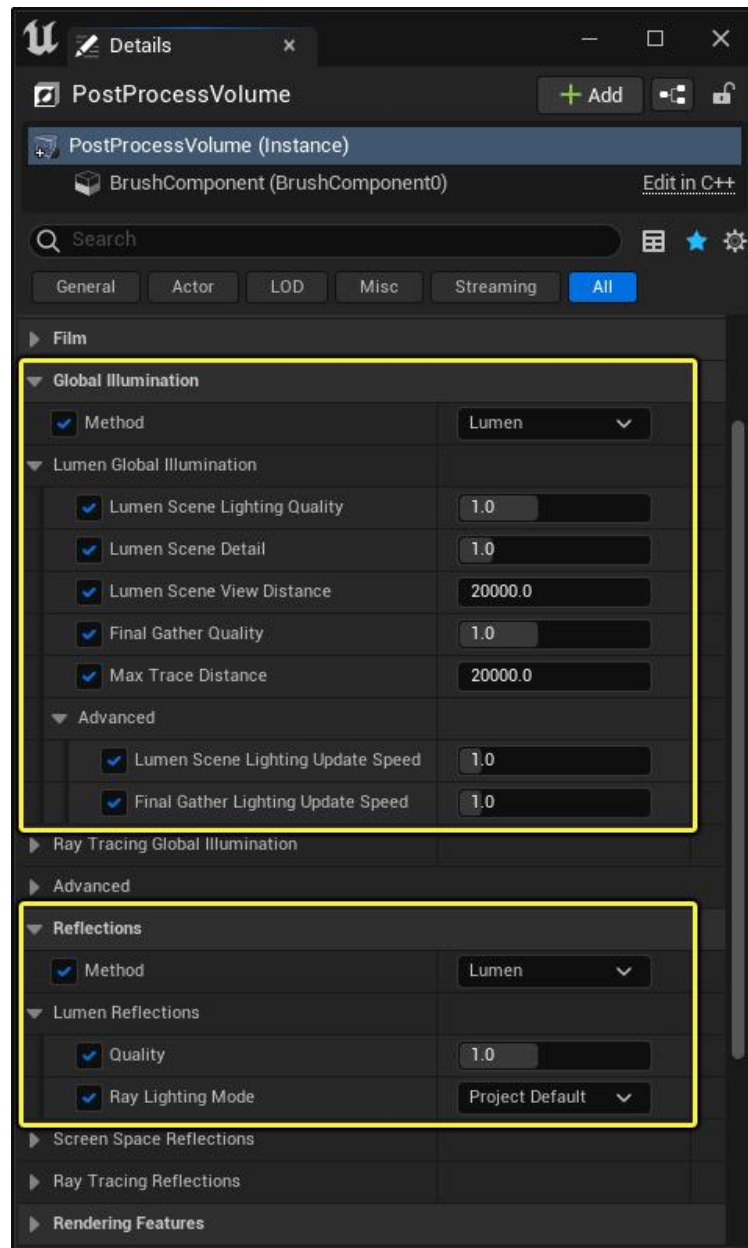


Figure 74: Lumen PostProcess settings

6. Economic Study

In this chapter we will study the economic impact of the material library development. We will consider factors such as human, software licenses, time, salary, hardware...

Regarding the time spent in the project, as an estimation I invested 175 hours of work from which a 50% would belong to the design and implementation, 20% testing the different materials as well as their iterations which would have taken a 15% of the time. 15% to the investigation of different techniques used.

If I were working as a Junior Technical Artist in a big game company, my salary would range between 18k€ and 25k€, according to job offers and "Glassdoor", which is a web page that hosts different information regarding jobs such as salary, opinion on various companies, interview processes...

I chose a medium salary of 21k€ for a Junior Technical Artist position.

A month salary of 1.750€ each month will cost the company around 2.264€ each month, the approximate difference, 514€, will be paid by the company to the state for different reasons such as social security and other taxes: unemployment (5.5%), common contingencies (23.6%), fogasa (0.2%), formation (0.6%) ..., this averages the amount to around 14,15 € per hour which we will use to calculate the costs of the development.

Task	Time in hours	Economic cost per hour
Design Implementation of the different materials	87,5	1238,13€
Testing the different materials at different situations	61,25	866,69€
Iterating over the materials	26.25	371,44€
Researching different techniques	26.25	371,44€
Total	175	2847,7€

We must also consider the cost of each component and material that was used to make this project and amortizing it by estimating the lifespan of each product.

	Cost in €	Project duration	Cost per month	Product lifespan	Total
High-end Desktop PC	1600 €	1,093	22,22€	6 years	24,29€
Laptop	1000 €	1,093	20,83€	4 years	22,77€
Two 2K Monitors	500 €	1,093	13,89€	3 years	15,18€
Misc.	150 €	1,093	2,5€	5 years	2,73€
Total	3250 €				64,97€

No special software license was needed for the development of this project.

Then, the total cost can be calculated as follows:

	Cost in €
Development	2.847,7 €
Materials used	64,97 €
Total	2.912,67 €

From a business perspective, this project can be doable due to the reutilization of the produced assets across different projects and being able to extract as much value of them as possible. However, as using these technologies and developing for them supposes a great initial investment it can be a risk some companies, especially smaller ones, would not be willing to make.

7. Results

This project aim was to have a raytracing system working in the game project, and as a consequence, gain knowledge related to the role of a Technical Artist within the game development industry. The material library has been built on top of technology and techniques designed by other people. That allowed me to focus on the creative aspect of it, to explore what was needed and what was more appealing.

The results of working in this project are a set of prototype materials that can be used within a game and can be easily expanded upon. These materials feature some of the newer technologies available in real time graphics, which can be enabled with a sublevel profile system that is left to finish.

The architecture that Epic games presented within Unreal Engine 5 to develop different assets such as materials, particle systems, material hierarchies and blends among others made the development easier as I did not have to worry about the most low-level things such as the display device registration within the graphics API, letting me focus on researching interesting techniques and how technologies worked underneath.

As we can see in the following images, as a simple performance test, I added a lot of simple 3D meshes with a glass material, my GPU, a RTX 3060, could handle more than a hundred entities without dropping from 60 frames per second.

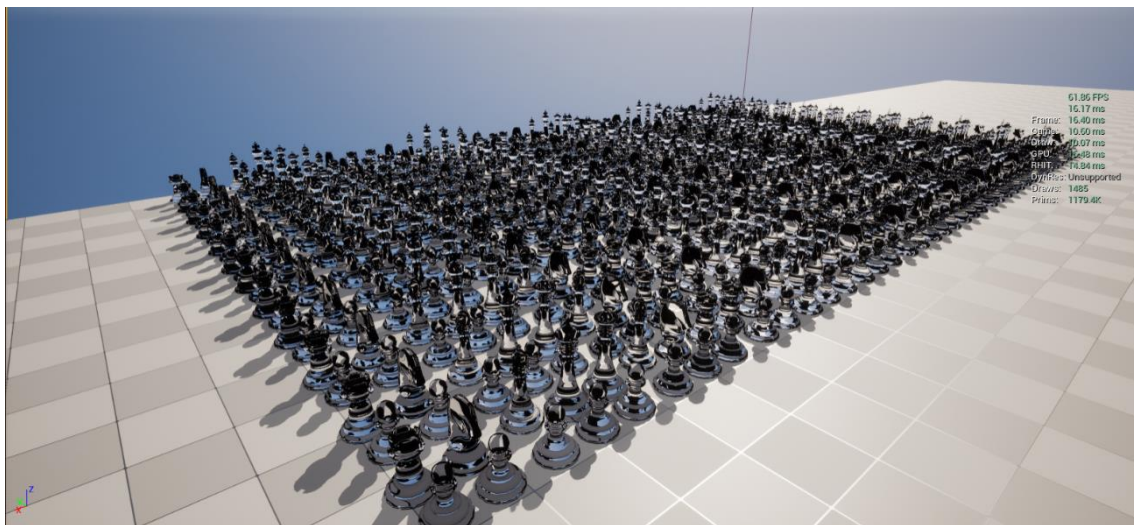


Figure 75: Simple 3D meshes with raytracing glass material rendered using a RTX 3060.

However, as we can see in the next image, when more complex meshes are added, the performance goes down exponentially. In our case, we added several of the Magic Chess 3D Pawn models with a raytracing clear glass material and the frames per second were quickly downed to less than thirty.

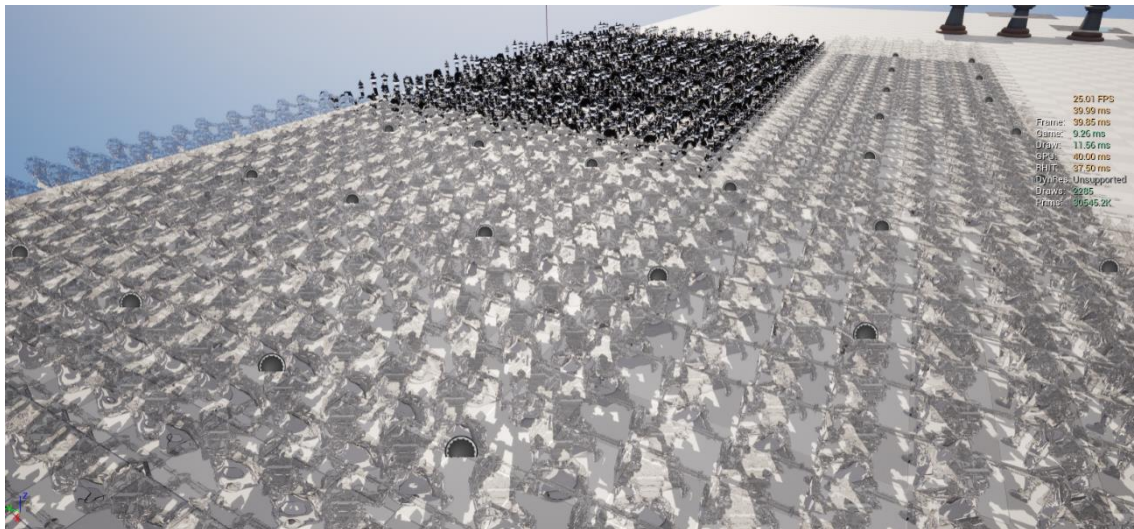


Figure 76: Complex 3D meshes with raytracing glass material rendered using a RTX 3060

Next up, we will see in the following images the huge visual difference we get when we disable Raytracing features in some comparison images. In the left part of the image, raytracing will be enabled, and in the left part of the image it will be disabled.

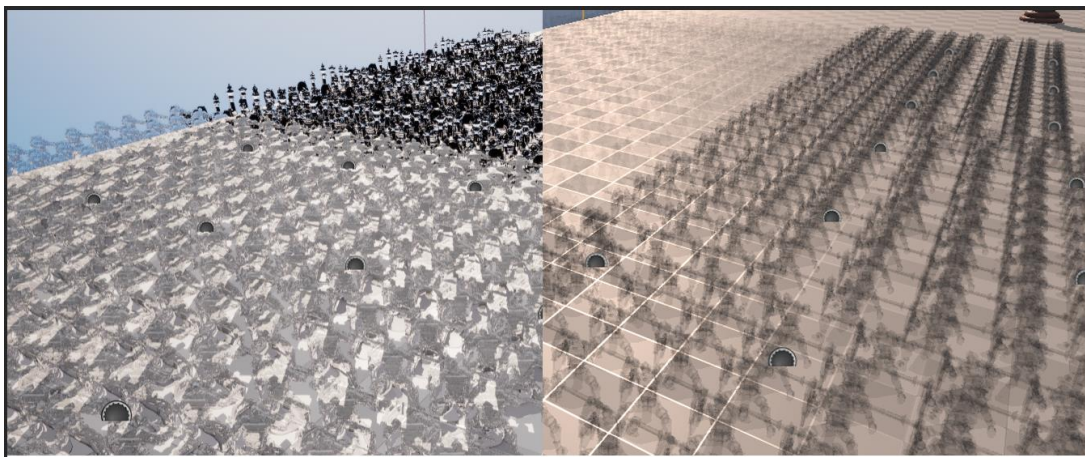


Figure 77: Pawn performance test RTX on/RTX off.

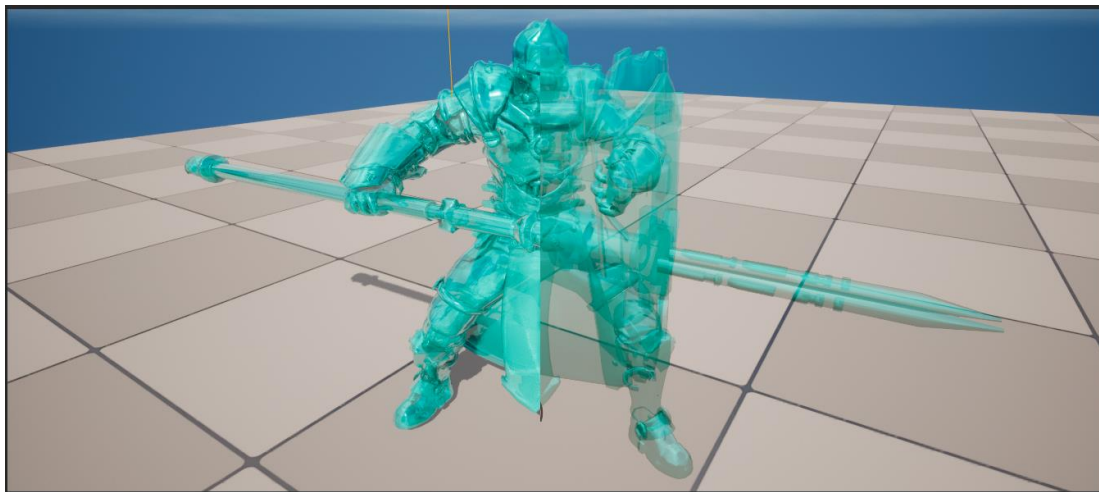


Figure 78: Pawn with translucent material RTX on/RTX off.



Figure 79: Pawn with Mirror material RTX on/RTX off.

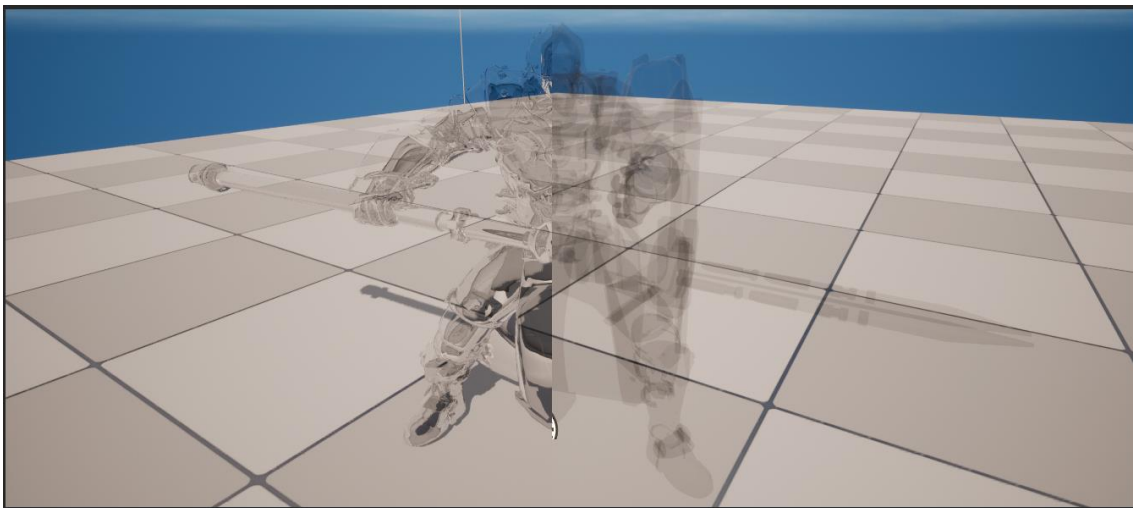


Figure 80: Pawn with Glass Material RTX on/RTX off.

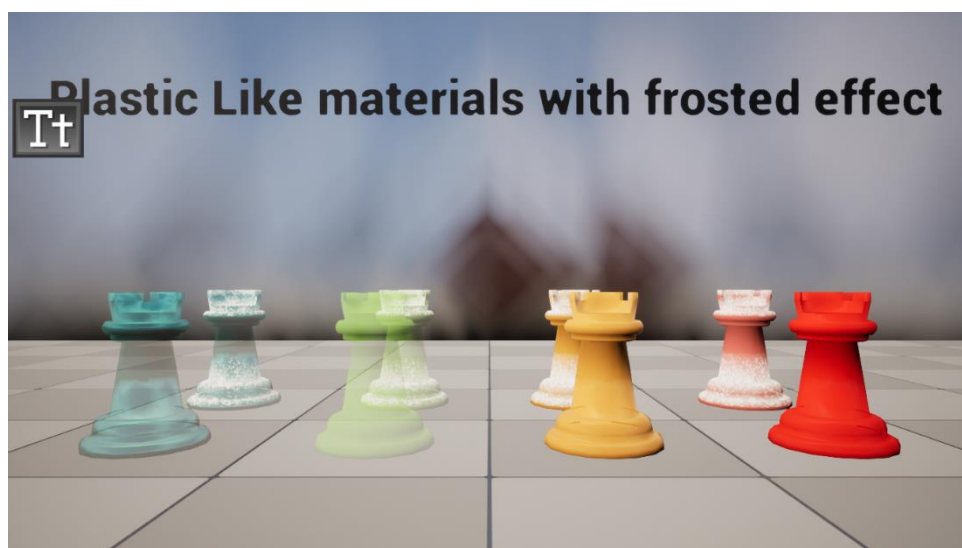


Figure 81: Rooks with materials and mirror on the back with RTX off.

To conclude, the material library and effects are still in development, which means there is room for improvement and changing things to enhance it, as it is not a system that has been in development for many years but a series of assets that conform a smaller architecture that is still expanding in the direction that the Magic Chess project needs point.

8. Conclusion and future work

All the Materials and Material Instances created for this project are into the new Magic Chess game project from Eclipse Games. I have worked in this material library in order to get this project published. My objective was mostly focused on the visual development of the project, specifically around the Materials development, that are being used in the different props. I have worked as part of a team of developers, each one a professional in his field and I am satisfied with the work done as of yet, even though it is still not completely finished.

This work presented satisfies fully the objectives that were set and advanced the development of the Magic Chess project on the visual aspect of it.

There are some things left to do such as extending more the material library and tweaking the different effects, as videogame development is an iterative process. Also finishing the different sublevels for the Raytracing configuration in different target hardware. Supporting more metallic materials would be the next step to take.

The Master Material system is set and can be extended further by adding more layers to the different materials that are already in, which would allow for more diversification in the visual aspect of the project.

9. Bibliography

Physically Based Rendering, From Theory to Implementation. By Matt Pharr, Wenzel Jakob, and Greg Humphreys.

Fundamentals of Computer Graphics. By Steve Marschner and Peter Shirley.

Real Time Rendering by Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Michal Iwanicki and Sébastien Hillaire.

History of 3D Rendering

<https://pixelperfect-studios.com/history-of-3d-rendering/>

Art of Video Games exhibition

<https://www.artofvideogames.org/>

Exemplification of Technical Artist job responsibilities

<https://www.qt.io/blog/technical-artist-job-description-roles-and-responsibilities>

Rory Jones Technical Artist blog

<http://www.roryjones.net/>

Unreal Engine 5 Material Documentation

<https://docs.unrealengine.com/5.0/en-US/unreal-engine-materials/>

Homogeneous coordinates

<https://www.tomdalling.com/blog/modern-opengl/explaining-homogenous-coordinates-and-projective-geometry/>

Shading and Light models

<https://cglearn.codelight.eu/pub/computer-graphics/shading-and-lighting>

Chess Games along the years

<https://gamerant.com/best-chess-video-games/>



10. Annexes

10.1. Project Proposal

0. Note:

The project proposal has been modified due to the project's game engine of choice changed and the workflow within the new one made possible work directly on the materials, which passed on to be the development objective of this project.

1. Project's Title

Realtime Raytracing capable Materials for videogames.

2. Description and Justification of the Topic

The project consists of the investigation e implementation of Materials that make use of the Raytracing in a commercial videogame developed in Unreal Engine 5, as well as some visual effects derived from the Raytracing capabilities. This technology is being used these days to offer a superior visual quality and reach more advanced light and water caustics simulations.

3. Project's Objective

- a. Getting to know the different effects such as reflections and refractions, as well as the state of the art in Realtime Raytracing as well as Game Graphics in relation to the proposed project.
- b. Create a robust architecture within the materials that allow for further iteration during the game development process.
- c. Create a series of basic materials that use Raytracing such as glass or plastic.
- d. Implement effects to those materials to create a visually appealing set of materials such as frosting, reflections, caustics...

4. Methodology

The methodology will be defined with the tutor in the reunions of the project.

5. Task Planification

The planning of the different tasks will be defined with the tutor in the reunions of the project.

6. Additional Observations

The tutor of the project will be Eduardo Jiménez Chapresto.

10.2. Meetings

Date	26-11-2021	Reunion	Online through Discord.		
Nº	1	Hour	10:00 AM	Duration	20 minutes
1	Quick review on the project and talk about different ideas for it.				
2	Talk about the most interesting Raytracing effects: Reflections and Refractions.				
3	Mention on writing methodologies and game engine.				
4	Greenlight from tutor to start researching and placing some prototype systems.				

Date	09-12-2021	Reunion	Online through Discord.		
Nº	2	Hour	10:00 AM	Duration	24 minutes.
1	Short talk regarding the work.				
2	Extended talk about research advancements and papers on Acceleration Structures.				
3	In depth talk about Compute Shaders and Tree-like acceleration structures.				
4	Set next steps towards the implementation of the base system in Unity.				

Date	10-02-2022	Reunion	Online through Discord.		
Nº	3	Hour	10:30 AM	Duration	30 minutes.
1	Talk about the state of the project.				
2	Showing current functionality in Unity, decision to swap to Unreal Engine.				
3	Talk about the steps we should make to port what we already have into the new engine.				

Date	31-03-2022	Reunion	Online through Discord.		
Nº	4	Hour	11:00 AM	Duration	25 minutes.
1	Quick talk about the project state. The port to UE5 was finished successfully.				
2	Comment on the UE5 Architecture and particularities.				
3	Talk about the differences in architecture encountered.				
4	Set next steps in the development process.				

Date	28-04-2022	Reunion	Online through Discord.		
Nº	5	Hour	10:30 PM	Duration	30 minutes.
1	Quick chat about how Magic Chess was going.				
2	Reminder on the state of this document.				
3	Extended talk about different features we would like to support.				

Date	08-08-2022	Reunion	Online through Discord.		
Nº	6	Hour	18:00 PM	Duration	25 minutes.
1	Short comment on the project state.				
2	Show of different features we are currently supporting.				
3	Comment on doing some profiling to fetch performance data on Raytracing.				
4	Talk about which GPUs support native hardware raytracing and which emulates it.				
5	Comment on this document review.				