UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Architectural Rendering and 3D Visualization

Germán Insua Perdomo

Director: David Sánchez Carreras

Degree: Videogame design and Development

Year: 2021-22

University: Universitat Politénica de Catalunya

Content Table

Summary

The following thesis, “Architectural Render and 3D Visualization,” describes the process of creating, rendering, and optimizing an Interior Design using a 3D Engine as the principal tool. The tool used during the development is “Unreal Engine,” which allows rendering and interaction in real-time with the scene.

At the end of the process, we can obtain an interactive scene rendered with high-quality materials trying to reach a realistic real-time scene by mixing modeling, texturing, and illumination techniques.

Furthermore, scripting is contemplated in the project scope, looking to optimize the environment where we will be developing the scene, and developing some tools.

Keywords

Rendering, Physical Based Rendering, Scene, Architecture, 3D Pipeline, Illumination, Scripting, Optimization, Unreal Engine, Post-processing.

Links

Github Project

Github Latest Release

Download Latest Release

Excel Budget Calculations

ArchViz Google Form

Drive Folder

Youtube Project Showreel

Table Index

Figure Index

Glossary

ArchViz: Architectural Visualization. The branch of 3D focused on the generation of architectural environments.

Frustum: Geometric shape sort of a pyramid that works as user field of view

HDRP: High-Definition Render Pipeline. Unity Package enables users to create through lights and materials realistic scenes.

nGons: Polygons that contain more than four vertex/edges.

PBR: Also known as *Physical Based Rendering,* is a rendering technique based on 3D light calculations.

Raytracing: Rendering technique that calculates the trajectory and path of rays released from a source point. These rays interact with the objects calculating and defining the physical properties.

VR: Virtual Reality. Technology that allows a user to make use of a head-mounted display to simulate and be immersed in different kinds of software.

VSync: Vertical Synchronization. Used to avoid screen tearing because GPU framerate and Screen refresh rate is not synchronized.

1. Introduction

1.1 Motivation

The motivation to carry out this project has been to show that I've capabilities to build up from scratch an entire realistic scene using a 3D engine that looks like it will lead in the industry for the next few years.

In addition, developing this project will show which skills I've with 3D environments and code optimizations.

Besides, it's also a personal technical challenge because I will force myself to learn a new tool and improve my knowledge and skills in that software.

1.2 Problem Statement

The main goal it's creating an architectural interior scene into a real-time application. This statement means that during the process, technical challenges related to rendering and performance must be considered.

Render and performance quality, are identified as expected problems to solve and balance because if some of both in the project have trouble at any point, this can lead the app to some behaviours such as:

- Visual artifacts generated by bad geometries, mesh face normal inverted, noise, lighting, and shadowing…
- Low frame rates that slow down the app.
- Memory leaks leading app to run-time exceptions.
- High workloads that GPU's memory budget can't afford due to complex, lighting-shadowing calculations, loading textures and unoptimized geometries.

Nowadays, multiple techniques that can be seen later in State of Art 2.0 in different areas of the production allows affording a viable solution for the points seen before.

The result of this project will be a desktop application running a realistic interior scene that can be used in different fields analyzed in 2.4 Market Analysis.

1.3 Thesis General Objectives

The main objective for this thesis will be creating from scratch, designing and modeling a whole architectural interior environment, achieving a realistic result looking for optimizations without significantly reducing render quality.

Furthermore, this interior will be visualized through a real-time application such as Unreal Engine, this means that an app will be deployed and the goal will be reaching the implementation of different camera configurations, setting static viewpoints, transitions, and movement around the scene. In addition, performance and optimizations need to be measured.

The project will be done using 3D software such as Autodesk Maya, Substance 3D and Unreal Engine, standards nowadays in the market. Using this software leads through the project to create basic architectural blueprints, modeling and texturizing assets /environments, programming code snippets and optimizations for a real-time execution app.

1.4 Thesis Specific Objectives

-Architectural Design: Plan and design a basic blueprint. This environment will be based on modern architectural interior references. In addition, an asset list needs to be ready for this objective.

-3D Prototype: Iterate the blueprint and furniture distribution with a blockout before starting the production phase. Implement this prototype into Unreal Engine to test with basic lighting and camera interactions that will require first code snippets and functionalities.

-**Modeling**: Modeling all the assets from the list previously planned and linked with real references.

First, all the assets/environment need to be modeled in scale avoiding high-poly geometries to prevent performance issues during the real-time rendering. Later the geometry needs to be optimized (deleting non-visible faces, loop reduction etc.) and each UV set need to be done to texturize the model later.

If the asset for some reason requires more detail, this can be done using Maya sculpting tools getting a High-poly version that will be baked later in Substance 3D when we are ready to texturize each item.

-Texturing: Texturize specific assets using Substance 3D and PBR materials. Textures need to be exported and transferred to Unreal Engine with the low-poly version of each model.

- Building Scene: Replace the blockout with the finished assets using the functional scene previously done. Furniture position iterations will be allowed trying to reach a satisfying result.

- Lighting and Post-processing: Lighting and adding post-processing effects in the final scene to accomplish realistic lighting.

- **Final Build**: Optimize the performance in the scene using profiling tools, testing, and deploying versions reaching the final version of the project.

1.5 Project Scope

The project goal requires high loads of work just for one person, that's why the scope is limited to performing a medium-sized interior scene that can be done in 3 months, avoiding looking for larger surfaces.

One limitation of this project is the null knowledge in architecture, which requires real-world references at the beginning to generate the blueprints and design.

The project result and the assets done to generate the app can be used in different scenarios such as video games, real state selling, museums, cultural heritage reconstruction and cinema as well.

Finally, the target that will take benefit of this project are 3D artists or developers that want to learn step by step how to build an interior architecture scene and know which techniques apply optimizing the performance to reach sustainable results.

2. State of the art

2.1 Modeling and Texturing

First, to develop the project, modeling techniques used nowadays must be considered to achieve 3D meshes that later need to be implemented in the scene. Before going deeper with techniques applied the concept of modeling needs to be defined.

Modeling is known as a process where real surfaces are recreated as 3-dimensional virtual objects generated with polygons. Some modeling techniques used nowadays are:

Polygon to Polygon

The common way to model. In this method, the user interacts with polygons, trying to imitate the surface that want to create. The most used interactions imply vertex-edge-face modifications with translations, rotations, scaling and extruding among the 3-dimensional axis (x, y, z).

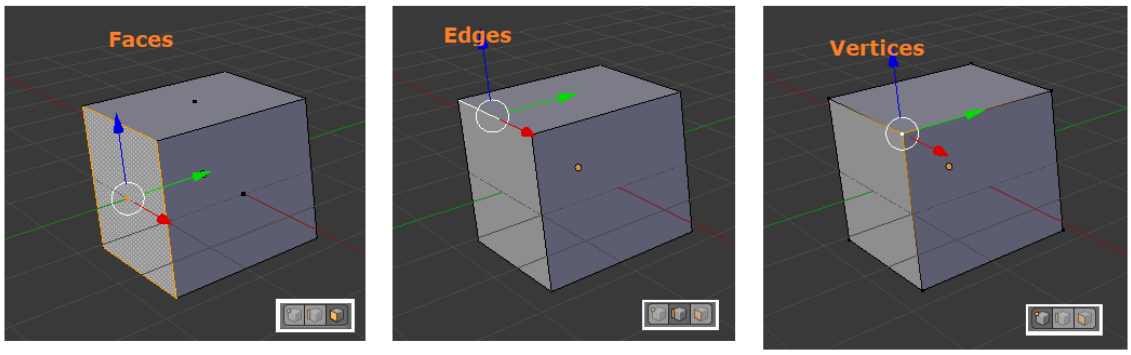

Fig. 1 Main interactions with polygons

Subdivision

Alternative technique to recreate surfaces. Through this technique, the user stars with a simpler or primitive geometry, subdividing and deforming the mesh iterating the object till achieving the result.

Moreover, adding or moving edges take a relevant role in this technique because allows the user to define the shapes before adding subdivisions.

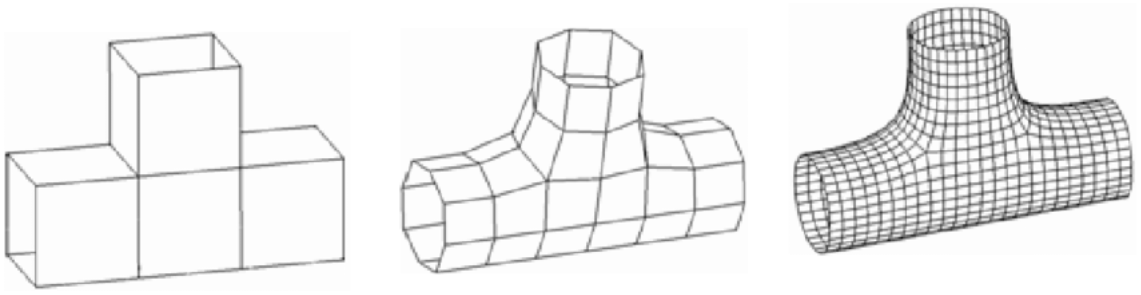

Fig. 2 Subdivision steps to generate shapes in a primitive mesh.

Nurbs

The foundations of modeling with NURBS are found in mathematics to achieve smooth shapes accurately.

Non-Uniform Rational B-Splines from now NURBS allows the user to create explicit, implicit, and parametric curves that can be modified in a set of points to generate the desired shape.

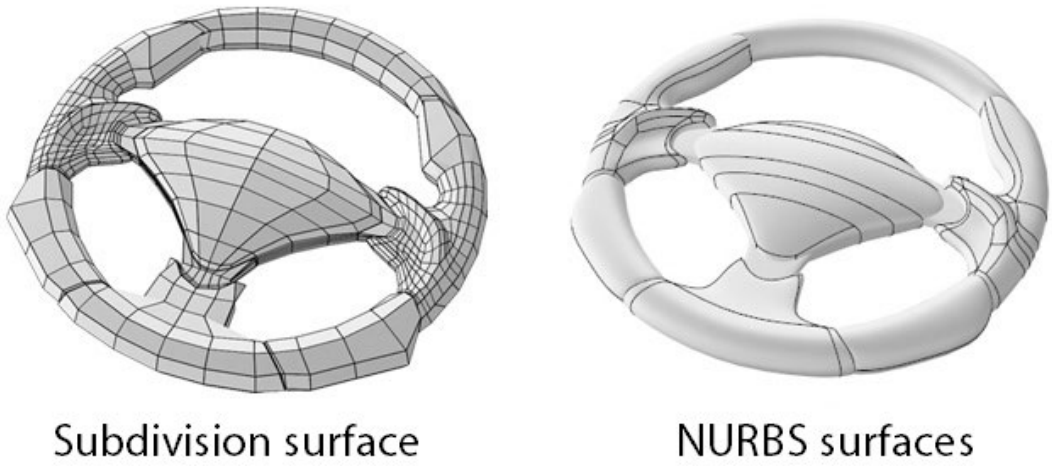

Fig. 3 Difference between subdivision and NURBS modeling techniques.

Sculpting

This modeling technique is most times used to generate organic meshes. This digital technique, it's the closest method to traditional sculpting with clay.

Commonly used by artists to generate organic characters, this technique also allows recreating artificial meshes that require more detail in their shapes.

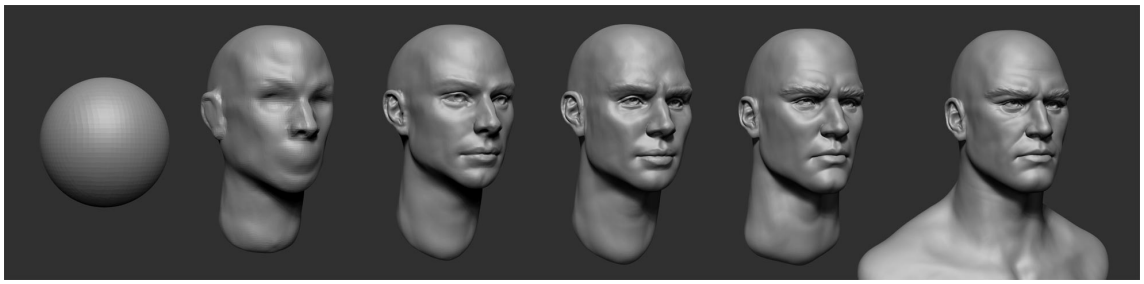

Fig. 4 Sculpting process from a base sphere.

Procedural

This technique allows users to generate shapes based on a set of rules or algorithms using recursive functions.

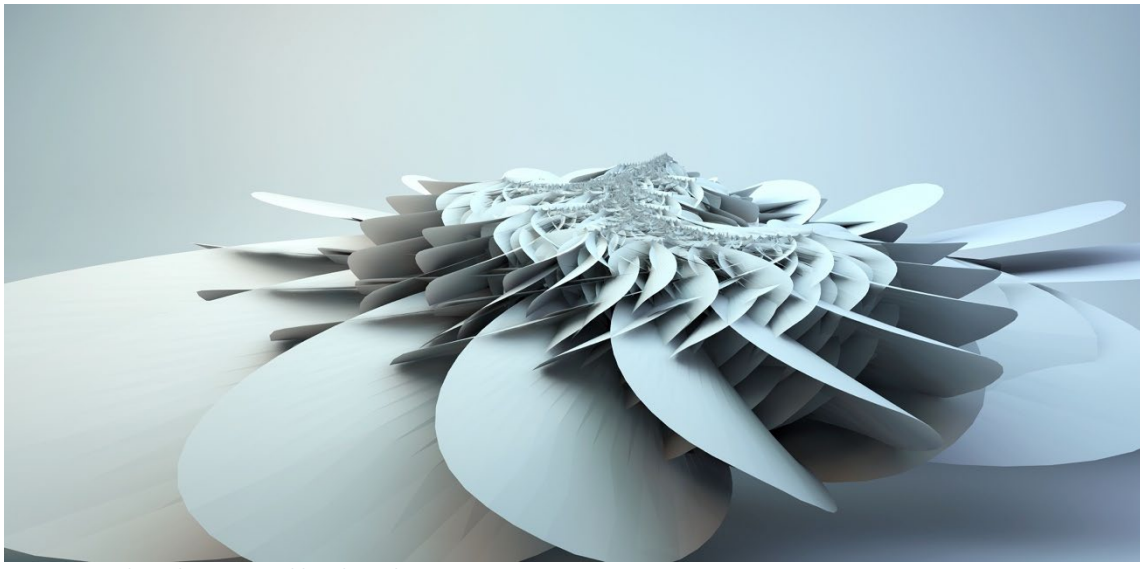

Fig. 5 Fractal mesh generated by algorithms.

Photogrammetry

Process of taking multiple photographs of an object from different angles to generate a 3D mesh based on the data gathered by the pictures. The generated model bears a strong resemblance to the real one and helped with great quality textures applied to the mesh achieved by the data collected. Photogrammetry can be done by applying Aerial Land or Satellite photos.

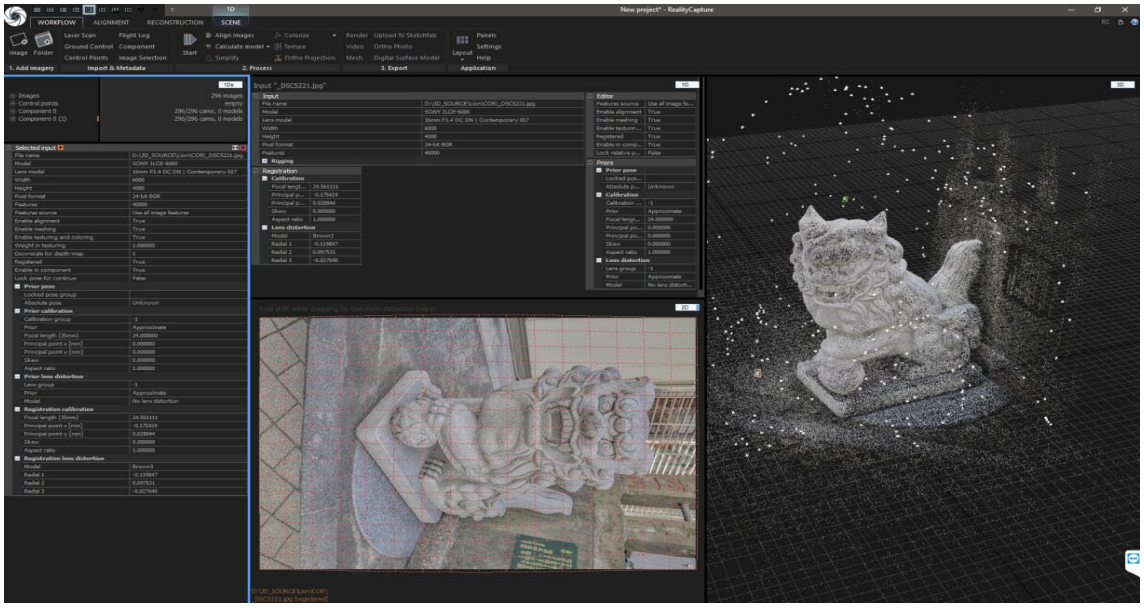

Fig. 6 Photogrammetry technique applied to a real object.

UV Mapping

Each 3-dimensional object generated by polygons can have a 2D representation via projection. This process it's also known as UV Mapping and with this relationship between points 3D and 2D coordinates geometries can be texturized.

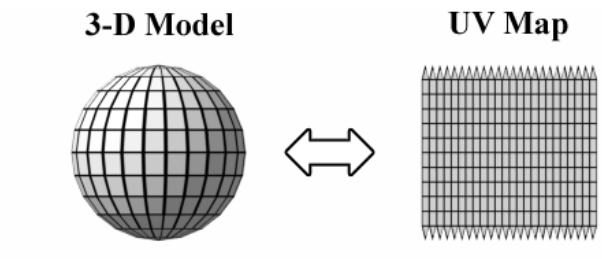

Fig. 7 3D Sphere projection into 2D coordinates.

Texturing

Afterwards, getting the UV mapping correctly done the next process in the pipeline it's texturing the model. Using the previously done UV maps the model can create different information such as diffuse/albedo, specular, roughness, normal maps…

Combining different maps can make the object achieve some physical properties important for render steps.

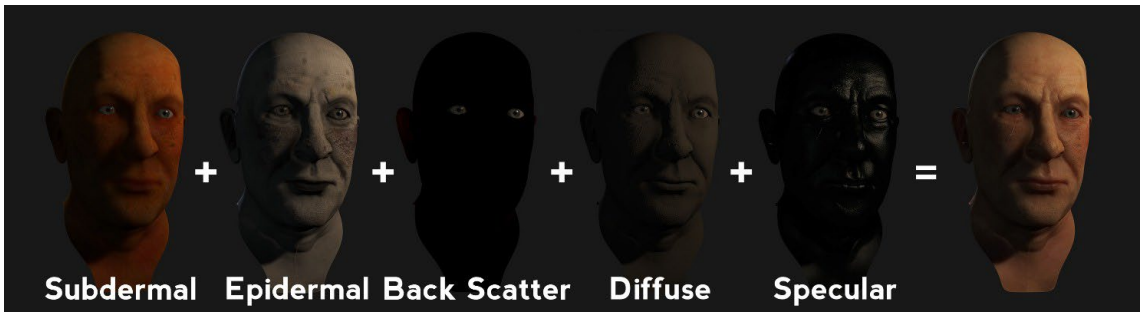

Fig. 8 Different render layers combined.

2.2 Illumination and Rendering

Global Illumination

Technique based in the use of algorithms that adds realistic lighting calculations to a 3D environment. Using this technique, the scene doesn't take in mind just the light source, nearby objects can reflect, refract, and generate shadows that will generate indirect illumination to the rest of the scene.

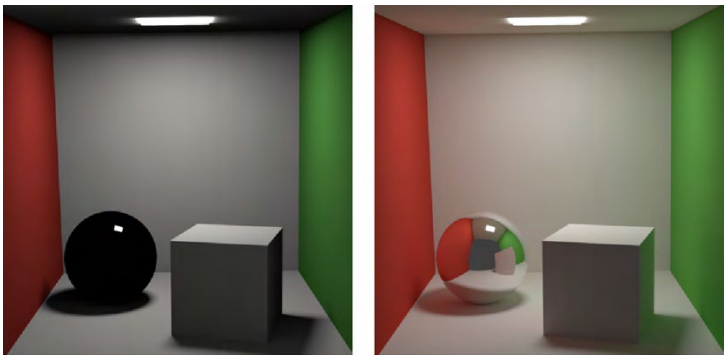

Fig. 9 Effect of the global illumination into a scene.

Volumetric Light

Technique used to generate light beams from a source. These beams are generated due to the calculations of the light source over with a 3-dimensional volume that simulates containing dust, smoke…

Fig. 10 Effect of a light source over a dust volume simulation.

Rendering

The process to obtain a final image from a 3-dimensional environment to simulate objects, lights, and materials. Rendering can be done in different ways, such as:

-Static Renders

Higher quality performance it's not trouble since the hardware is powerful nowadays, with great results with no limitations more than the time.

-Real-Time rendering

Afford quality in real-time renders requires multiple optimizations due to the hardware's need to render multiple frames each second.

Fig. 11 Result of an interior environment render.

Post-Processing

Group of effects applied at the end of each frame, modifying the result of the render to obtain higher quality in each frame.

Fig. 12 Effects of applying post-processing filters.

2.3 Optimizations

Frustum Culling

Rendering technique that allows optimizing the performance avoiding render calculations of non-visible objects in the scene using the view frustum as bounds for the geometries in the scene.

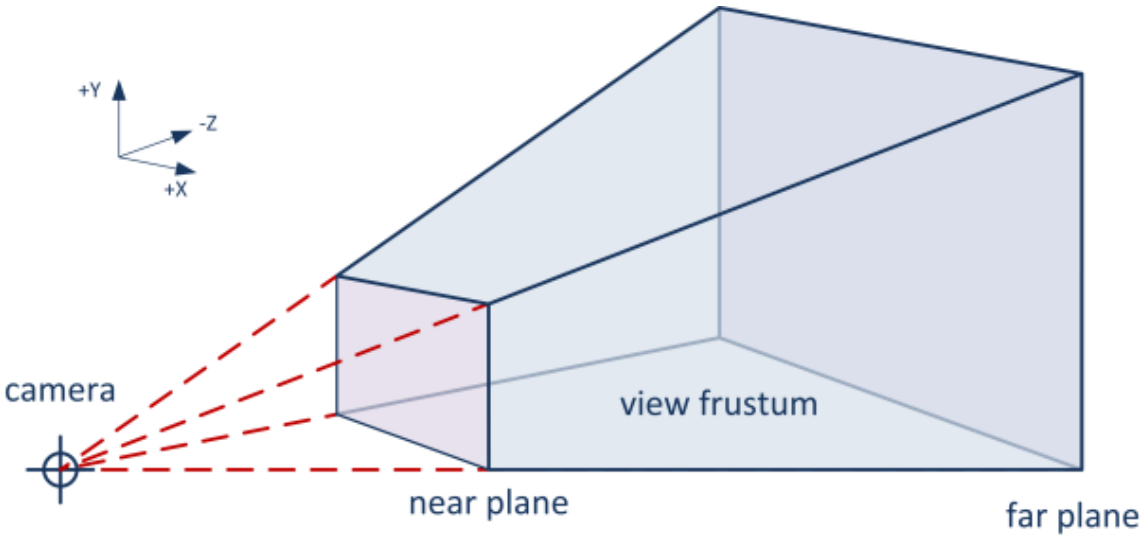

Fig. 13 Frustum culling.

Light Baking

Illumination alternative that prevents the use of static lights in real-time increasing the render performance for each frame. It's a good alternative for static scenes because stores all the lighting information into light maps for of each object.

Otherwise, if any object in the scene it's dynamic, Blend Probes can store light information in any section of the scene that will affect just to dynamic elements.

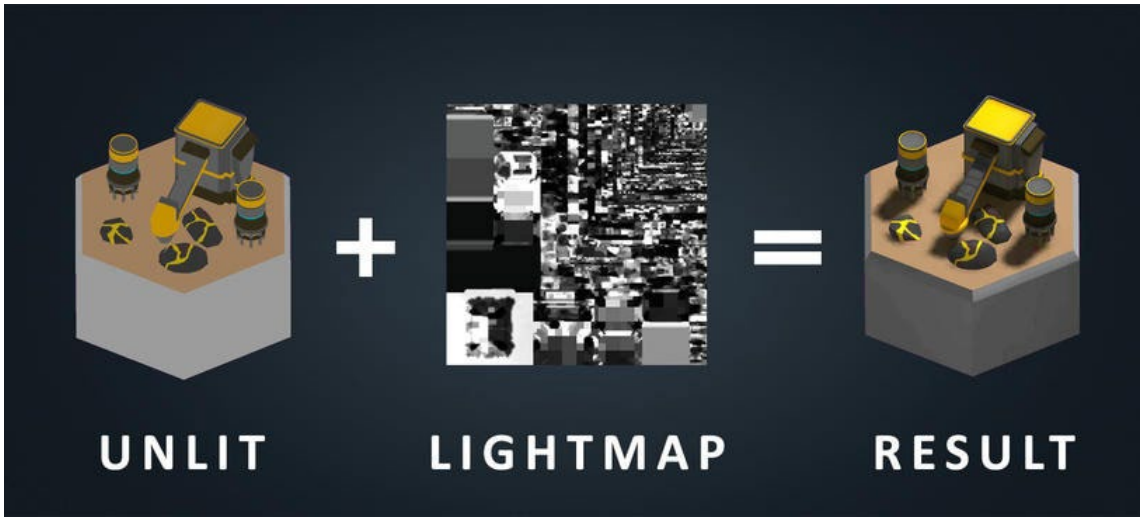

Fig. 14 Light information pre-calculated into a map and applied to the object.

Texture Baking

This technique allows low-poly objects to preserve quality and detail by using information from the high-poly geometry. This information it's transferred through different texture maps to be used by the low-poly version.

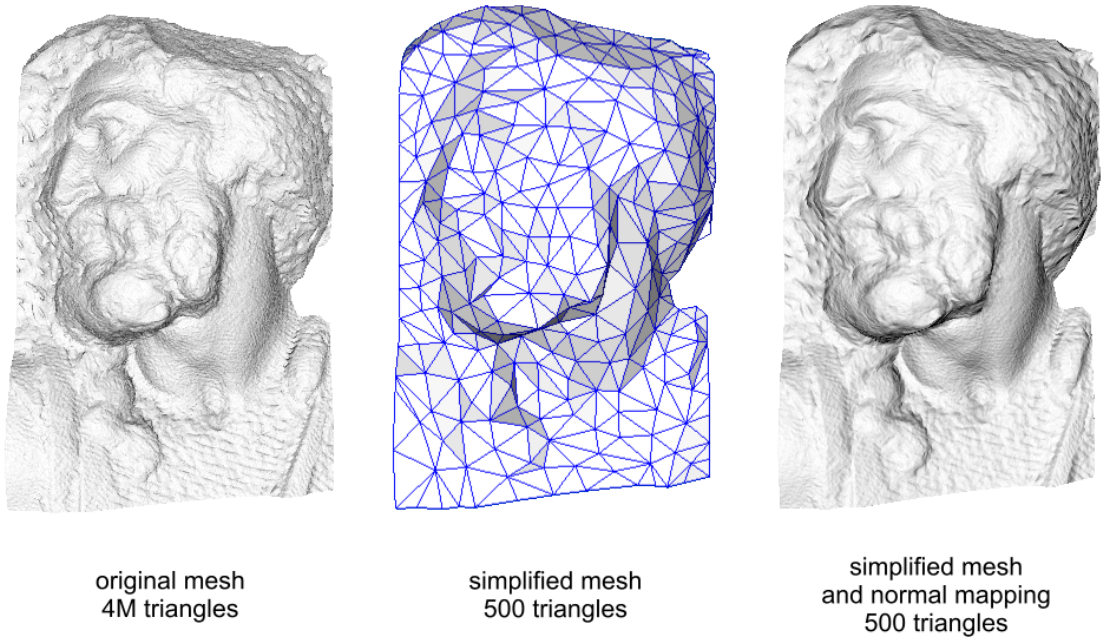

Fig. 15 High-Poly mesh detail transferred to low-poly version through bake technique.

LODS

Known as Level of Detail. Used to increase rendering performance by swapping the mesh based on the distance between the mesh and the camera. If the camera is close to the geometry, the quality will be higher because it requires more detail. Otherwise, the mesh it's replaced with lower-quality geometries.

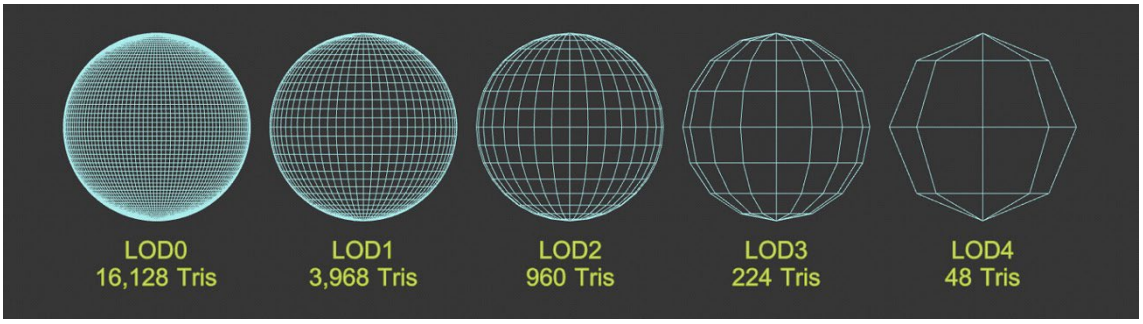

Fig. 16 Sphere mesh LOD's example.

Atlas Texture

Alternative to prevent massive texture loading into RAM memory. All the UV sets from different objects (assets with different meshes, grouped by small assets…) are set in a big canvas called Atlas that will store the same information than n objects losing some resolution through the process.

Fig. 17 House atlas texture with different mesh parts grouped in the sameUV map.

2.4 Market analysis

Nowadays, the project can be fitted into different market products.

The closest option to our project can be found in video game development, where a real-time rendering it's a must-have and need to apply multiple techniques to optimize the performance of different types of hardware (Personal Computers, Consoles, Mobile.).

In addition, all the resources of hardware intended for video games are destined for rendering, logic, and AI… therefore the task of just recreating 3D environments in real-time requires less hardware budget. Assets and environments need to be optimized but nowadays videogames can't reach 100% visually satisfying results due to

limitations with the hardware.

Fig. 18 Videogame: The Last of Us II. Interior Environment.

Other market that can be related with this project can be found in the real state sector via new housing. However, real state tends to avoid real-time applications in favor of static renders o pre-rendered animations due the higher quality that can be achieved with these techniques that doesn't require rendering multiple times per second.

Fig. 19 Architectural Visualization. Static render from an interior.

Finally, in the world of cinema through CGI, excellent results are obtained, but everything is pre-rendered and not shown in real-time. In addition, this sector tends to work with different static render layers in post-production.

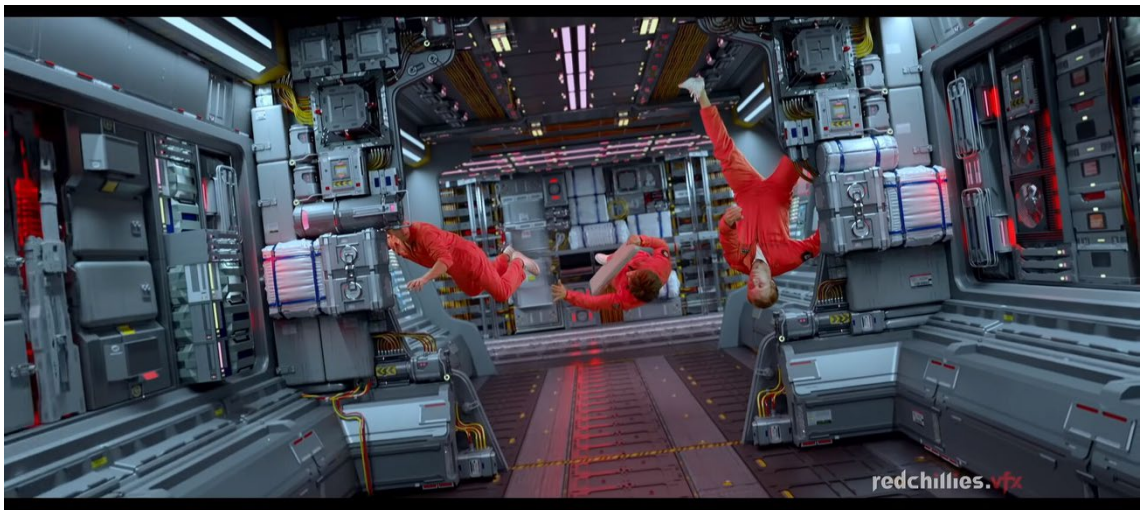

Fig. 20 Spaceship Interior CGI.

2.5 Unreal Engine

Unreal Engine it's a 3D graphic engine developed by Epic Games. Nowadays this engine can hold different types of productions including game development, architecture, film and production, product design etc.

Unreal it has become more relevant in the industry during the recent years due to some powerful features that Epic Games have developed and implemented in their engine.

Recently on 5th April 2022 Epic Games officially released their 5.0 version of the engine. This release includes some features useful for the current project such as:

Nanite Virtualized Geometry

Nanite technology allows users to create geometries with massive polygon details using an internal mesh format to render pixel scale detail and high object counts.

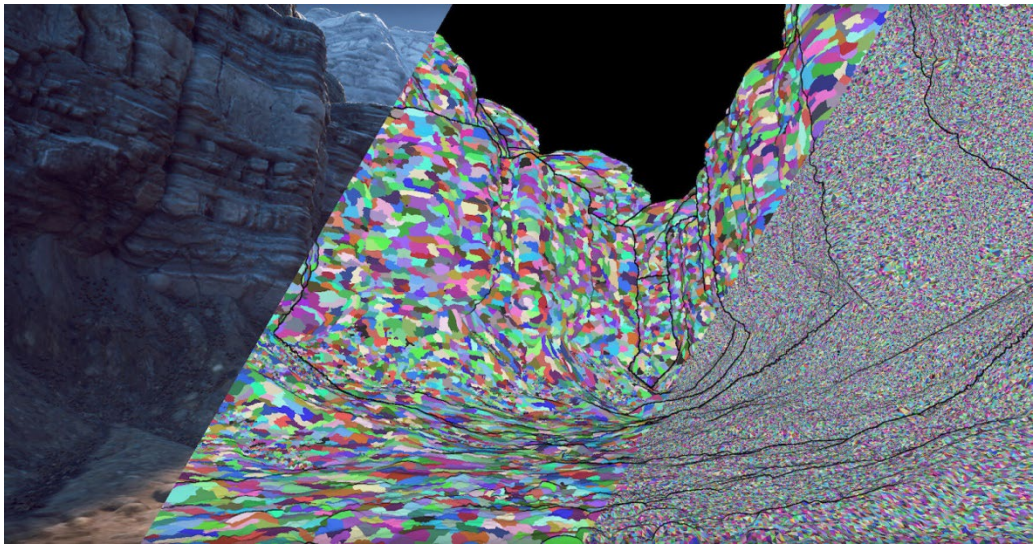

Fig. 21 High-poly mesh using Nanite tris and clusters.

This technology, optimize the mesh size by reducing and compressing data, improving the performance related to the memory budget in the scene while the meshes are being rendered.

Useful technology to create high detailed meshes without applying optimizations such as texture baking and occlusion culling to increase the render performance due to memory budget, polycount, mesh draw calls…

Lumen Global Illumination and Reflections

Lumen is a global illumination system that works in real-time using software ray tracing instead hardware raytracing.

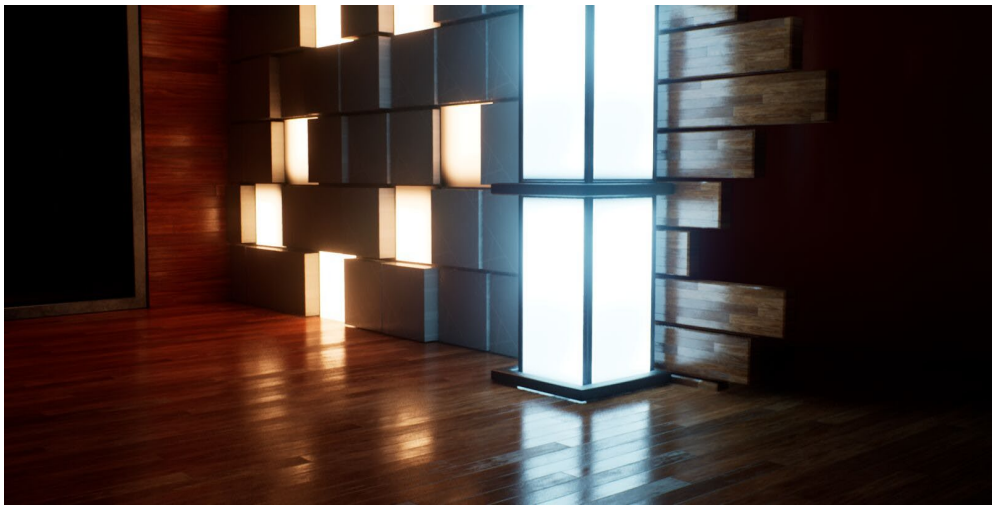

Fig. 22 Source of light generate indirect light reflections.

Lumen offers more dynamic scenes using lighting going one step further and leaving old static optimizations such as light baking allowing non-RTX GPUs to create more realistic environments.

3. Project Management

3.1 Methods and Tools to track project workflow

During the project, a well-known agile method such as **Kanban** have been used.

Furthermore, the **Gantt** chart has been used too just to have a larger scope of the project through the different production phases.

Although it's a one-man project this agile method has been chosen for its versatility because it allows to:

-Track each task

-Avoid task overload

-Flexibility

-Entrust product quality

3.1.1 Gantt

The main purpose of using a Gantt chart in the project it's just oriented to place milestones for each production phase in which Kanban Agile method will take control. Gantt chart only will allow having a bigger scope of the project through the time.

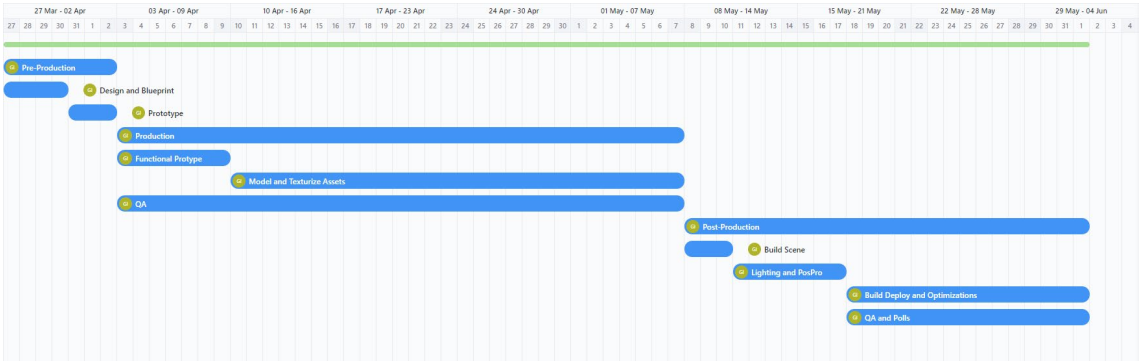

Fig. 23 Gantt char planning with main stages and sub-tasks.

As seen in the Gantt char the project follows 3 production plan phases (seen later in 4.0 Methodology). The production stages are:

-Pre-Production: March 27th,2022 – April 2nd,2022

-Production: April 3rd,2022 – May 7th,2022

-Post-Production: May 8th,2022 – June 1st,2022

In addition, each main phase contains the subtasks to be developed and led by the Kanban method board.

3.1.2 Jira

The main tool where the project will be managed. Jira will be used to track each task using a Kanban method board where tasks can be created, moved, and edited through different states for each production stage.

In addition, Jira allows ensuring the quality of each asset produced with its reporting issues tool. Furthermore, the project needs to be able to test and track possible issues for each version released in the real-time app.

Fig. 24 Kanban agile method board example.

3.1.3 Github

Although code it's not the main workforce of the project, Github entrust to store the project using version control Git and releasing the built app.

3.2 Validation Tools

Throughout the project, different validation tools have been used to ensure the quality of the project and track possible issues.
As seen before in 3.1.2 Jira tools ensure to track the quality of each asset produced using the Kanban board. Before considering an asset completed each one needs to pass its own developed QA test that measures:

-Mesh: Polygon quantity, topology, uvs, and scale.

-Textures: File quality and size.

Furthermore, for each app deployment, Github allows reporting issues through the released versions tracking bugs. Reports can be fulfilled with images, descriptions, and states for each reported issue.

Moreover, external google forms polls in the final stage to technical profiles with knowledge in 3D or developing skills can validate the performance gathering user information such as:

-Device specs: CPU, GPU, RAM…

-Performance budget/stats: Framerate, CPU workforce, GPU/RAM usage.

-Bugs: Report sheet filled by the users just in case they found issues or non-expected behaviors.

In addition, the poll can be exploited to extract relevant information related to the result asking users about artistic aspects such as Asset quality, scene realism, illumination, and design…

3.3. SWOT

	Helpful	Harmful
Internal Origin	**Strengths** -Complete knowledge of 3D pipeline and work environments -Interest in archViz.	**Weaknesses** -Unreal Engine tool non-mastered -Null experience in architecture -Partial time work for the project
External Origin	**Opportunities** -Lack of architectural visualizations in real time. -Learn more about 3D optimizations -Increase my skills as Artist/Developer	**Threats** -Multiple techniques nowadays to achieve similar objectives -Lot of competitors using alternatives

Table 1 SWOT Analysis.

3.4. Risks and contingency plans

Possible risks identified in the project, and their corresponding solutions sorted from lowest to highest:

Risk	Solution
Less Experience working with UE	Transfer the project to Unity using HDRP
Mesh polycount decrease app performance	Generate LOD for each geometry complex asset
Weird Performance behaviors	Create different graphic settings
Lack of assets in the scene	Color and shape variations from already generated assets

Table 2 Risks and contingency plans.

3.5. Budget initial analysis

Disclaimer: This is a project for educational purposes and it's not intended to be a for-profit product, which has repercussions in many areas of budget calculation.

However, and to be more precise, all the calculations and results of this section have been obtained in a context where the result of the project it's going to be a commissioned job as Freelance. This must be taken into consideration for the next results.

Before going deeper into the budget calculations, the project requires defining which things will be **tangible/intangible costs**.

[The following explanation can be found Links Page 5. – Excel Budget Calculations]

3.5.1 Intangible Costs

Production time costs

Time it's the most valuable resource of the project need to have mind of. To value this intangible resource, junior roles needed to develop this project have been compared with current market rates in Spain. The roles involved to develop this project are:

- Junior 3D Artist [8€/h] - (*Glassdoor - Junior 3D Artist Salary*, s. f.)
- Junior Developer [11€/h] – (*Glassdoor - Junior Developer Salary*, s. f.)

With the previous average market rates for these roles in Spain extracted from Glassdoor and following the Gantt and tasks planned for this project the total expense equals to:

Role	Wage/H	Work Hours	Total cost	
Junior 3D Artist	8,00€	144	1.152,00€	**Total Salaries**
Junior Developer	11,00€	156	1.716,00€	2.868,00€

Table 3 Salaries per role.

Freelance costs

As freelance/individual persons the local governments and laws require to pay an autonomous monthly fee to perform the jobs. In this practice case, the project requires a newbie professional, that's why the fee is lower for the first 12 months, from 294€ to 60€.

Freelance Cost	Fee*	Months	Total
Autonomous Fee	60€	3	180€

Table 4 Freelance costs.

*** Lower fee first 12 months**

Software license

Due to our methodology model, it's defined by a pre-production production and post-production pattern, we need to define specifically how much time of production we are going to need and define a threshold value in time to avoid planning miscalculations.

This need to be defined because in the production stage we will need to acquire the proper licenses to start creating our content and optimize the production costs during the whole project.

Software to acquire:

Software	License Cost*	Months Acquired	Cost per Month	
Autodesk Maya 2022	279€	1	279€	
Pixologic ZBrush 2022	36,45€	1	36,45€	
Adobe Substance 3D	18,19€	1	18,19€	**Total Software**
Unreal Engine**	0€	3	0€	334€

Table 5 Software costs.

*** Monthly Subscription payment**

**** Free till overcome the benefit threshold**

Although acquiring payment software during one-month looks hopeful, this has been fitted in the project planning as seen before in 3.1.1 over the Gantt chart.

The production stage starts April 3rd,2022, and finishes May 7th, 2022 (34 days) where modeling and texturing tasks have been planned to require a software subscription for 27 days to create assets to optimize the budget.

Otherwise, if the original plan changes, the worst scenario for the project related to software budget calculations means acquiring 2 months of payment software leads the software costs to 668€.

3.5.2 Tangible Costs

Through this section, calculations will be based on the hardware and equipment unique payment and monthly workplace rental and supply costs during the project.

Utilities	Total	Months Acquired	Total Cost	Amortization*	
Hardware	925,36€	Unique pay	925,36€	74,03€	10 years at 20%
Equipment	130,00€	Unique pay	130,00€	5,85€	20 years at 10%
Rent Wage	350€	3	1.050€	~	**Total Utilities**
Supplies	130€	3	390€	~	2.495,36€

Table 6 Tangible costs.

*** Based in linear amortizations given by "Agencia Tributaria".**

3.5.3 Total Costs

To sum up this section with the previous Tangible and Intangible costs the project will require the value of 5877€ to balance the expenses of running out this project. To get a profit from the result of this project to the total has been applied a 15% commission and 21% taxes, obtaining the following result:

Total Costs	5.877€	
Total Profit (15%)	6.759€	**Revenue**
Total After Taxes	8.178€	2.301€

Table 7 Budget summary.

3.6. Planning Deviation (13/05/2022)

3.6.1 Deviation Justification

As seen in State of Art 2.5, the official release of Unreal 5.0 on 5th April 2022 forced the project planning to be modified to work with the latest stable version of the engine. This change has been made from Unreal 4.27.2 to 5.0.1 for benefit of the project, despite modifying the plan for a few reasons:

1. New Lumen technology implies the possibility of having real-time global illumination without ray-tracing video cards. Lumen will increase the quality of lighting and rendering in real-time.
2. Nanite technology optimizes each mesh with its own format, increasing the app performance and removing old traditional optimizations, e.g., occlusion culling

In addition to the previous reasons, Unreal allows the project to maintain the same time development reducing workforce-related with optimizations. Unreal ensures the optimizations of the app automatically doing some tasks such as:

1. Cameras are ready by default to apply frustum and occlusion culling for each mesh. Mesh dependent if the asset does have nanite enabled/disabled, switching between both optimization modes.
2. Instanced rendering of meshes with different transform values, e.g., walls/floor modules.
3. Runtime virtual texturing method supports large texture resolutions, optimizing the needed memory budget for each texture.
4. LODs for each mesh are generated when importing new assets.

3.6.2 Gantt reschedule

With the previous features, the GANTT planning has been updated taking in mind the engine release date and a margin of a few days to set up, read, and learn about these features that can be used in the project.

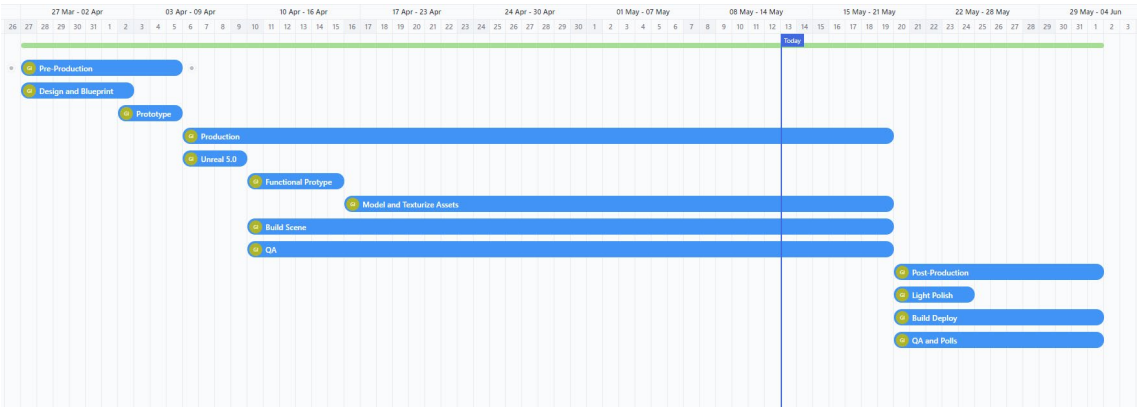

Fig. 25 Gantt chart reschedule.

In the previous image, the GANTT chart has been rescheduled to fit in the same original project duration. Some tasks have been changed during the methodology which will be explained later in Methodology Deviation 4.2.

3.6.3 Budget Update

These planning modifications imply a small budget change. Furthermore, “ZBrush” software has been removed to be used in the project, using instead “Autodesk Maya” sculpting tools and physics system.

The production phase now does have a duration of 1 month and 13 days. As explained before in Intangible Costs 3.5.1 related to software acquisition in the worst case we need to buy 2 months of software to run out the project leading the budget to:

Software	License Cost*	Months Acquired	Cost per Month	
Autodesk Maya 2022	279€	2	558€	
Pixologic ZBrush 2022	36,45€	0	0€	
Adobe Substance 3D	18,19€	2	36,38	**Total Software**
Unreal Engine**	0€	3	0€	594€

Table 8 Updated software costs.

Taking the previous total software cost and updating the budget sheet with the same development time and roles the total cost will lead to:

Total Costs	6138€	
Total Profit (15%)	7058€	**Revenue**
Total After Taxes	8.541€	2.403€

Table 9 Updated final costs.

3.7. Planning Deviation (01/06/2022)

3.7.1 Deviation Justification

As scheduled, the original plan has defined the project milestone date for **1st June 2022**. In the first approach, this delivery date looks optimistic, but the original plan contemplated **4,5 working journeys** as can be seen in the following table.

ECTS	Hours/ECTS	Total (hours)	Work journey	Days	Duration (Months)	Project Start	Project End
12	25	300	4,5	66,66666667	2,150537634	27/03/2022	01/06/2022

Table 10 Original project time distribution.

Due to several tasks and parallel projects aside from the university, the delivery date needs to be changed. This change will only involve **time distribution** (300 hours) along the days. Furthermore, budget calculations are not directly involved because the project will use the **same amount** of hours just **displaced in time**. This change can be achieved reducing the working journeys to balance the dates in time as can be seen in the next table.

ECTS	Hours/ECTS	Total (hours)	Work journey	Days	Duration (Months)	Project Start	Project End
12	25	300	3,25	92,30769231	2,977667494	27/03/2022	27/06/2022

Table 11 Updated time distribution.

As explained before, the working journey just changed from 4:30 hours to 3:15 hours. This distribution enables flexible development over the time compatible with other activities. Delivery date have been fixed to **27th June 2022** without the need of retargeting tasks or modifying budget costs*.

*Budgets are neither affected in monthly software acquisitions or tangible costs, the project budget sheet contemplated from the beginning a 3-month period for rent wage and supplies. Software in worst cases was contemplated for a 2-month period as seen in 3.5 Budget Initial Analysis.

3.7.2 Gantt reschedule

Following the previous dates, the Gantt planning has been rescheduled.

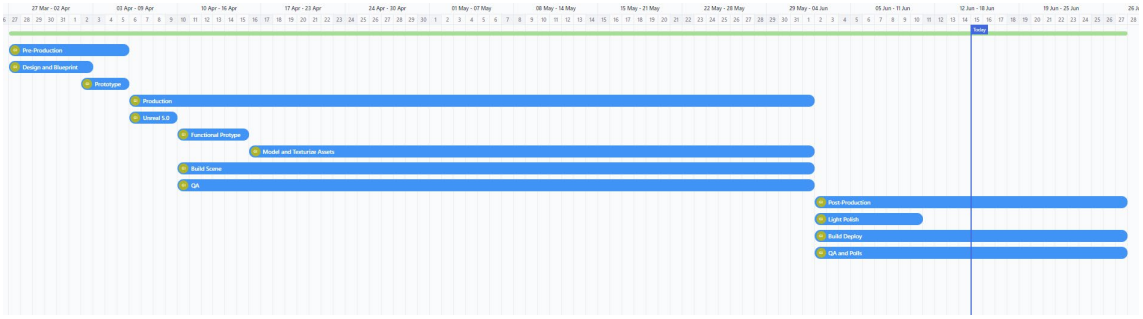

Fig. 26 Gantt char reschedule update.

The Gantt chart shows the same tasks as the last project update. Tasks are just distributed over more days and haven't been changed, which means that methodology neither changed.

To sum up with deviations, the new delivery date gives a brief margin from 27th to 30th June to set up the final delivery. This 3-day period is not contemplated in the project development.

4. Methodology

4.1. Methodology plan

The methodology applied follows a pre-production production and post-production model. This model has been chosen for a few reasons:

Firstly, the result of this project it's a real-time application which means that software needs to be deployed. The project can't start developing and deploying software since day 0 without content planned and ready before.

Despite the user interacts with the developed ap, rendering a real-time architectural environment, the project will follow this methodology instead user-centered design because the result just supports low skilled level interactions with the user (e.g., User press Keypad 1 to change camera spot).

Moreover, related to the previous point investigation and material its required before starting. A preproduction stage fits perfectly in the current project. Same scenario with the final stage in post-production adding the last few details and deploying different version through the QA processes.

In addition, this model it's a good way to limit the project scope for each production stage. This allows the project development to save resources such as time and budget, optimizing each task as seen in the Budget Initial Analysis (3.5).

-Pre-Production

Inside this stage, the workforce will be focused on planning. Ask what will need the project to be developed. Things such as references, blueprints, first prototypes, and asset list need to be done and well defined through this stage before starting production.

-Production

This phase will require a hard workforce.

Through the production, the project needs to have developed a functional prototype previously iterated in the Pre-Production phase with basic behaviors. Moreover, the task of modeling starts here, models done in Autodesk Maya will be transferred to create a High-Poly version in ZBrush. If the asset requires detail otherwise will be texturized in Adobe Substance 3D.

Besides, the quality of each 3D asset has been evaluated with a report sheet waiting for confirmation from the Kanban board to be done. Through this stage, the asset has been iterated and polished if required.

-Post-Production

During this final stage and with production already done, the functional prototype has been replaced with the assets produced into Unreal Engine.

The lighting and post-production take the control in the next step trying to achieve a realistic result.

Moreover, the different app versions have been released to be tested by users that will fill external polls, allowing the project to gather information and help optimizations to increase and adjust the performance.

4.2. Methodology Deviation (13/05/2022)

The pre-production, production, and post-production planning remain active, just with a few task modifications. Following the new GANTT chart and taking advantage of Unreal automatic optimizations, the following production stages changed:

-Production

Now during the production plan, the finished assets have been placed into the scene building the scene meanwhile assets are done. As before, the same concerning dynamic lighting. Moreover, due to the release of Unreal 5.0 between the already production phase a few research days were needed.

-Post-Production

Post-production now can remove tasks done in production such as lighting and building the scene from the original plan.

5. Project Development

5.1 Architecture and Design

First, the project needs to define some constraints and guides before starting the development and design. This step has been thought to prevent the limitations and the scope of the project.

In addition, these constraints also allow the project to avoid nonsense designs such as creating a bathroom with big windows in a block of flats or placing a chimney inside.

Constrains

One-floor apartment (avoid high ceilings)

1 Bedroom, 1 Toilet, Kitchen/Dining Room + Living Room

Some vegetation allowed (just inside)

Wooden floor and some wood walls it's a must-have

Some plasterboard ceilings with concavities.

Big windows around the living room and bedroom

With the previous constraints and guidelines in mind, the project requires getting some references and examples to build the bases of the scene due to the unknowledge of architectural concepts, as seen in 3.3 SWOT Analysis.

To reach that objective, the "Pinterest" website it's an ideal tool to get some references from visual designs following the guidelines. Furthermore, "Pinterest" allows pinning the references to create mood boards that can define the look of the project.

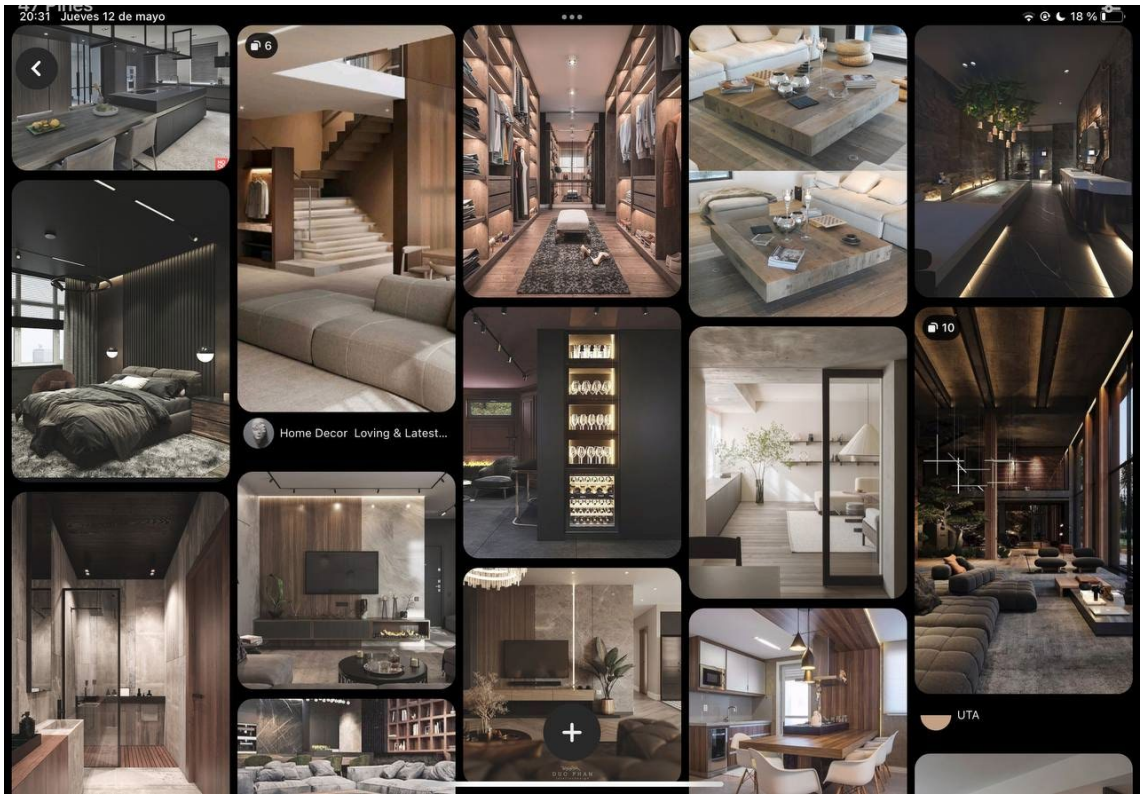

Fig. 27 Pinterest generated mood-board.

The mood board, previously done allows us to generate color palettes that have been extracted using different reference images and the website “Adobe Color”.

5.2 Blockout

One limitation in the project it’s the null experience doing architecture and floor planning. That’s the reason why the next step with which I’m more comfortable is prototyping, doing blockouts in a 3D software such as “AutoDesk Maya”.

Scales are an important part of the project because the lighting will affect in different ways depending on the size of the meshes, therefore Maya’s working units have been switched by default from cm to meters.

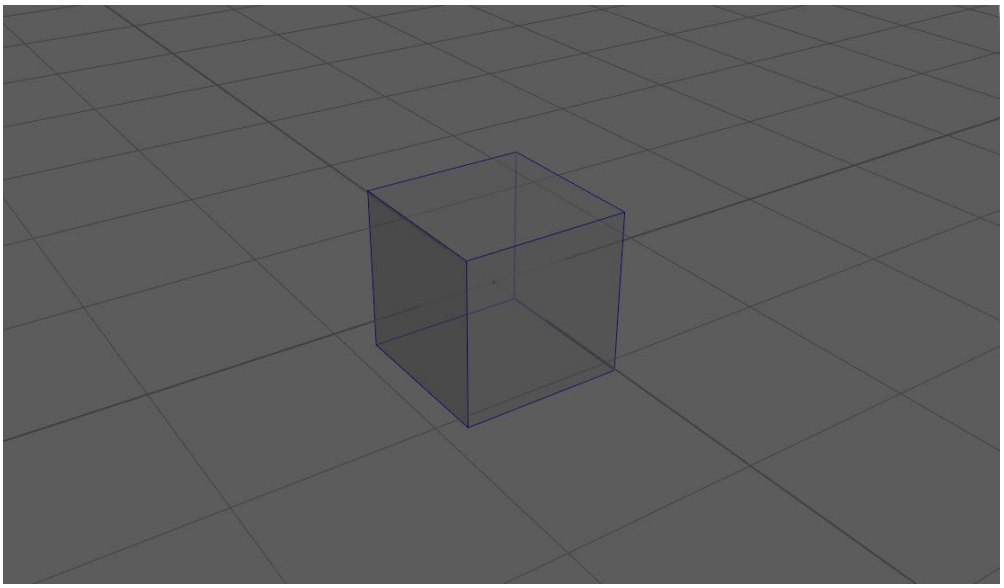

Fig. 28 Cube in scale to the working units 1x1x1 meters.

Some blocks did get approximated sizes to real furniture (e.g.: beds, sofas, and chairs) to reach realistic sizes between the different areas of the floor plan. The references and mood-boards did guide the blockout to create different areas in the flat, achieving the given guidelines.

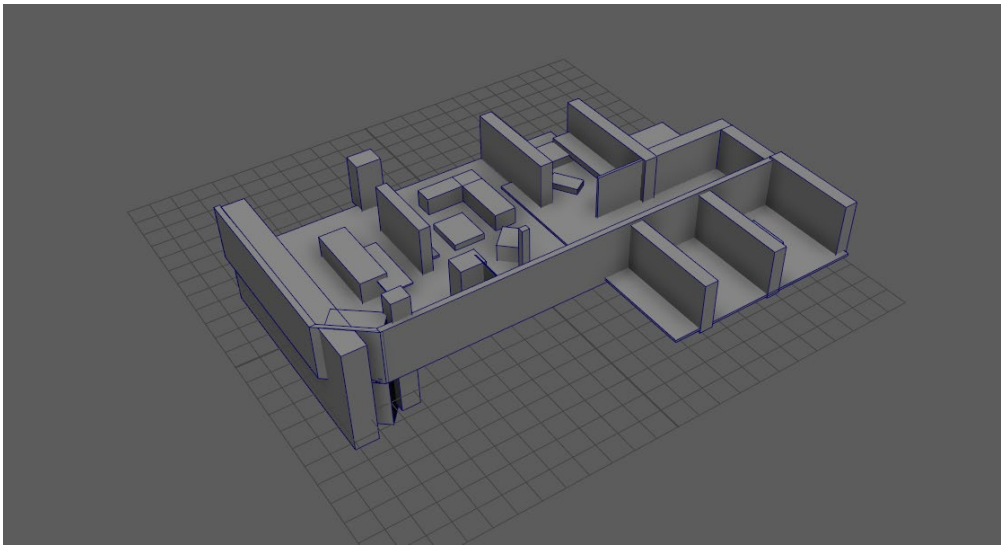

Fig. 29 Blockout in Maya.

5.3 Floor Plan Blueprint

Going one step back and with the blockout done getting the desired result based on the references the next step, it's creating the floor plan blueprint based on the top view of the blockout.

This task needs to be done in a more precise way because will contain the information about wall sizes and the whole surface sizes in m2. "FloorPlanner" free website helped to achieve this task conventionally.

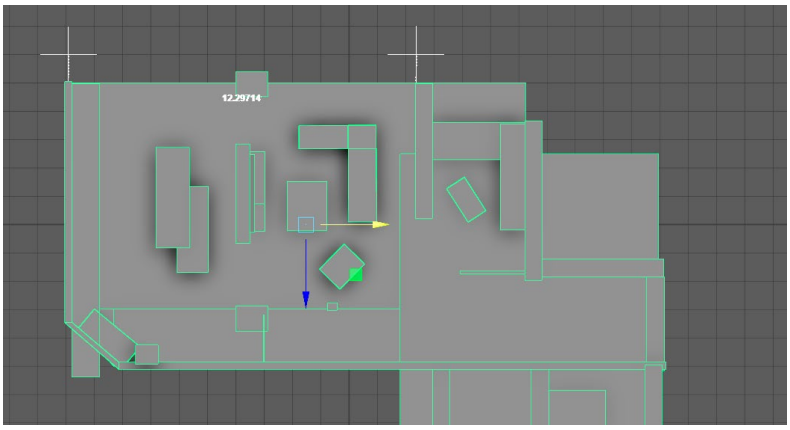

Fig. 30 Blockout projecting top view from Maya.

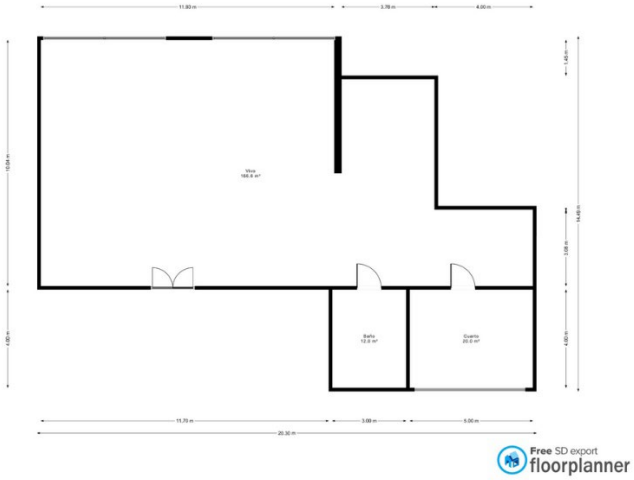

Fig. 31 Final FloorPlan based in the blockout.

The blockout has been projected in top view using Maya's measure tools to fulfill the floor plan into the blueprint, fixing and rounding some sizes. e.g., 12.324 meters from Maya's measure to 12 meters into the blueprint.

5.4 Prototype

Taking advantage of the blockout previously done with similar sizes to the real planning, we can achieve the first look in real-time using Unreal Engine with basic lighting and camera movements through the scene.

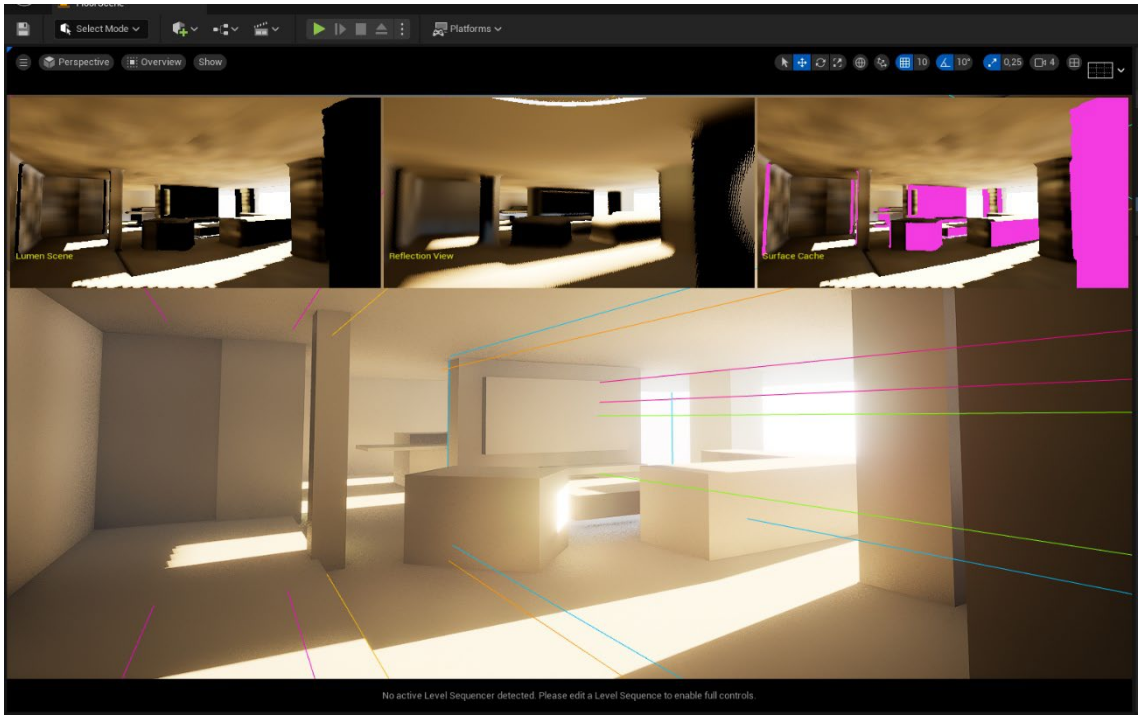

Fig. 32 Blockout placed in Unreal Engine.

5.5 Asset List

Finally, to sum up, with preproduction planning an asset list needs to be done. This list has been done using Excel sheets containing asset information such as nomenclature, current state photo/link to the real reference, approximated size, priority…

Those information fields have been thought in the fact of some assets can be done faster than others (e.g. a wooden desk doesn't require the same detail as a whole bed with all sheets etc.)

Moreover, some assets don't require being textured and just need to create a specific shader/material in Unreal to be rendered (e.g., ceramic in bathrooms, metallic elements as sinks, glasses from windows…).

Area	Asset_Name	ID	Nomenclature	Priority	LOD	Texturing	TextureSize	Polycount(tris)	Current Status	Reference
Kitchen	Sink	3	Ktch_Sink_03	Medium	LOW	No	N/A	290	Finished	/images_ae/p
Living Room	TV	5	Lr_TV_05	Medium	LOW	Yes	1024mb	64	Finished	/qled-8k-tvs/
Kitchen	Induction_Hob	3	Ktch_InductionHob_03	High	LOW	Yes	1024mb	18	Finished	m/images/I/6
Bedroom	Bed	9	Br_Bed_09	High	HIGH	Yes	N/A	N/A	Planned	hDfjFoeEUQq
Living Room	GoogleHome	5	Lr_GoogleH_05	Low	LOW	Yes	512mb	222	Finished	89gsVIG_3Z9c

Fig. 33 Asset list and QA.

In addition, this asset list will be useful during the production stage to track each asset and ensure the quality of each asset produced.

5.6 Functional Prototype

Using the basic prototype, the next step it's creating the basic user functionalities planned. To reach that objective, the main level blueprint has been modified and scripting has been developed using nodes in a more visual way to interact with code.

Because I'm not familiar with blueprint coding, the first step implies converting the desired interactions into pseudocode to transfer the logic to the blueprint.

```
/*CAMERA SWITCH SCRIPT 1st interaction*/

int cameraIndex = 0;
std::vec CameraVec;

CameraVec.pushback(*cameraRef1);
CameraVec.pushback(*cameraRefN);

if(Key.PRESSED == KeyCode.1)
    cameraIndex++;

SetTargetView = CameraVec[cameraIndex].GetCameraView();

/*FREE CAMERA 2nd interaction*/
if(Key.PRESSED == KeyCode.2)
    SetTargetView = Player.GetPlayerView();

/*CAMERA SEQUENCE*/

Animation cameraSeq = CreateCameraSequence();

if(Key.PRESSED == KeyCode.3)
    cameraSeq.PlayOnce();
```

Fig. 34 Pseudocode to generate blueprints.

As seen in the previous figure the first interaction is based on the iteration between multiple static actors in the scene(cameras) that will render the actor view.

The references of these static actors have been stored in an array storing each asset information that allows iterating the array elements. When the event of pressing the right arrow, it's executed the script iterates one position of the array to change the current actor and calls a function retargeting the new bind with the view transitioning the views with a blending.

Furthermore, to avoid array overflow the blueprint needs to declare a condition to reset the index that will iterate. In the following figure, we have a 3-element array storing cameras.

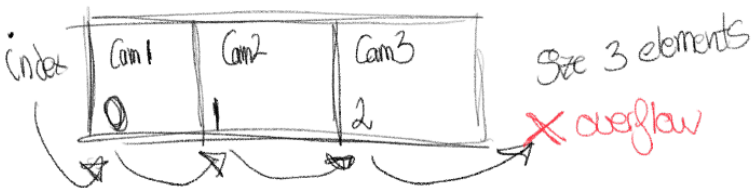

Fig. 35 Vector of cameras. Index can't iterate longer than array bounds.

This array needs to be iterated with the n-1 index being n the number of elements stored e.g., 3 elements mean [0,1,2] index. Moreover, the blueprint needs to reset that value to 0 looping through the array each time the max index has been reached (2).

This condition prevents reading non-allocated memory from non-existing index in the array further than 2.

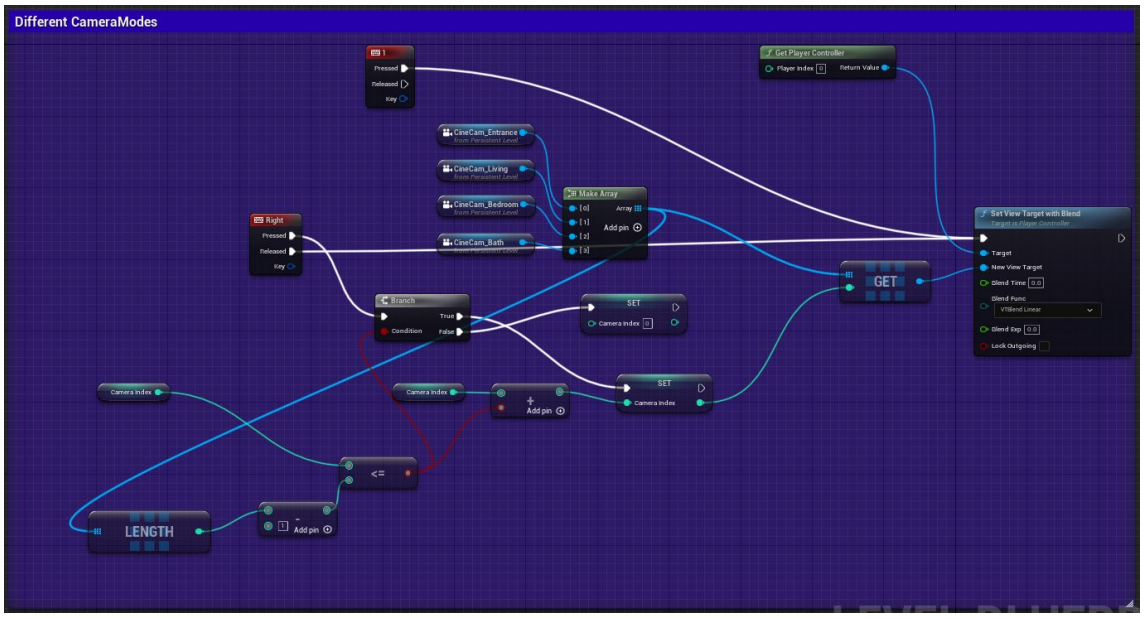

Fig. 36 1st Blueprint. Static Camera switch.

The second interaction fits the event of pressing key number 2. Through this event, the script will get the player position previously spawned at the beginning of the scene.

This actor will interact with the level, enabling free movement for the user and enabling collision with the current assets in the scene.

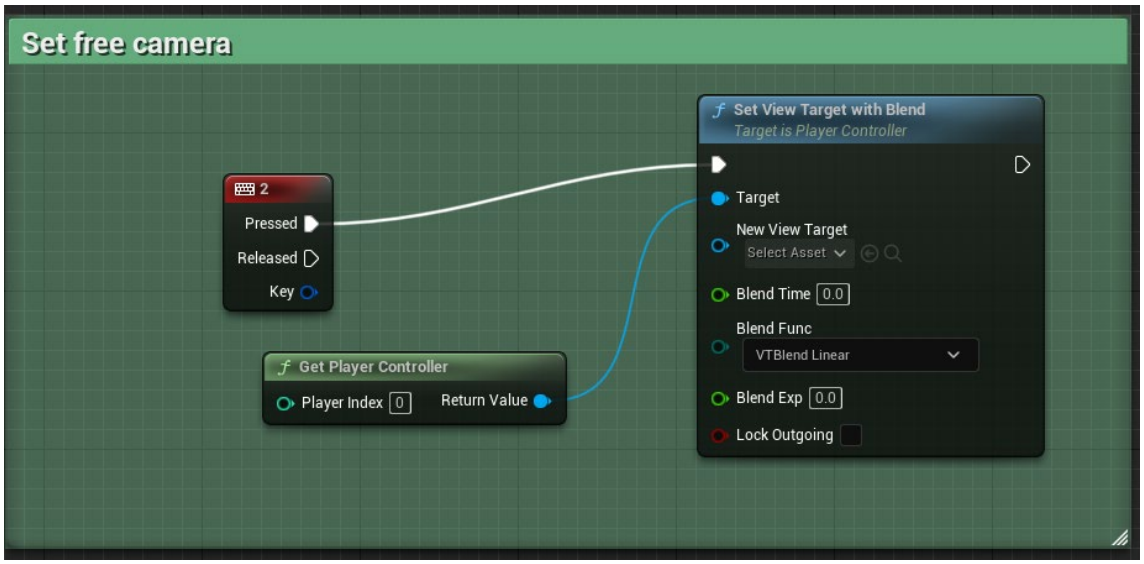

Fig. 37 2nd blueprint. Free Camera.

The last interaction combines the use of cameras and sequences with the execution of blueprints. Firstly, before enabling the interaction with an event in a blueprint the project needs to create camera sequences using the current cameras or create a new one with this purpose.

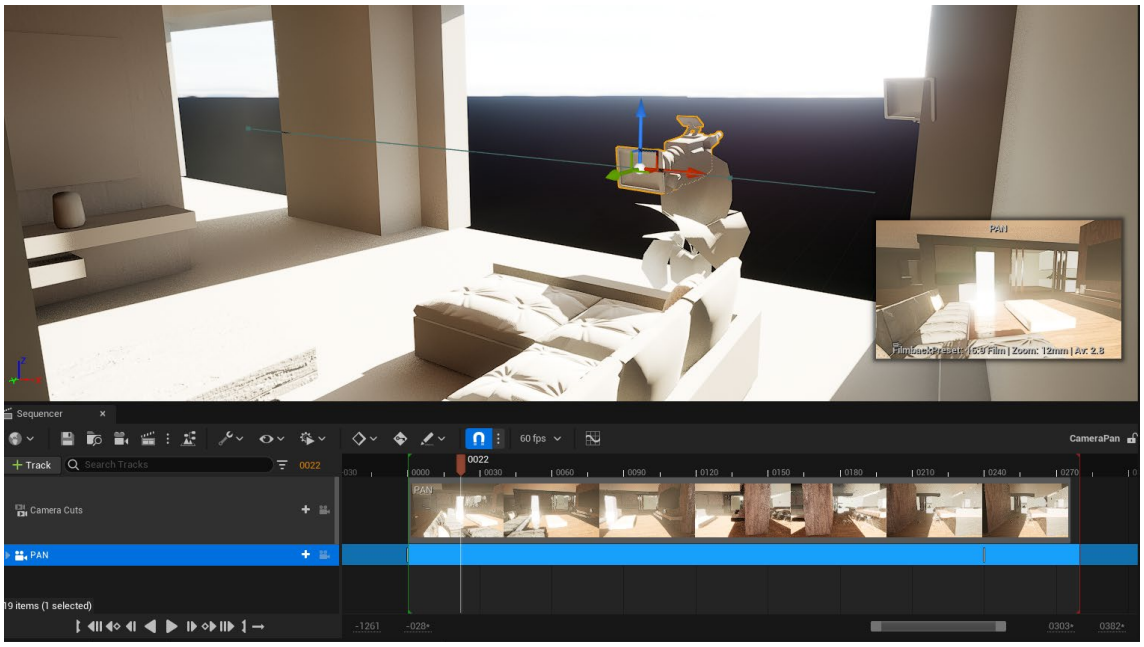

Fig. 38 Sequencer window with camera "Pan" linked.

Once the project has created the sequencer, a camera actor can be settled with different keyframes during the timeline storing information such as transforms, aperture, and exposition… (e.g. If a light has been placed in a sequence this actor can modify data as light intensity, color…). The timeline will interpolate the values between the different keyframes generating the animation.

In addition, these interpolations between keyframes create animation curves that can be modified. In the previous example, the curves have been modified to a linear form to perform constant speed through the axis that affects the camera movement “X” and “Z”.

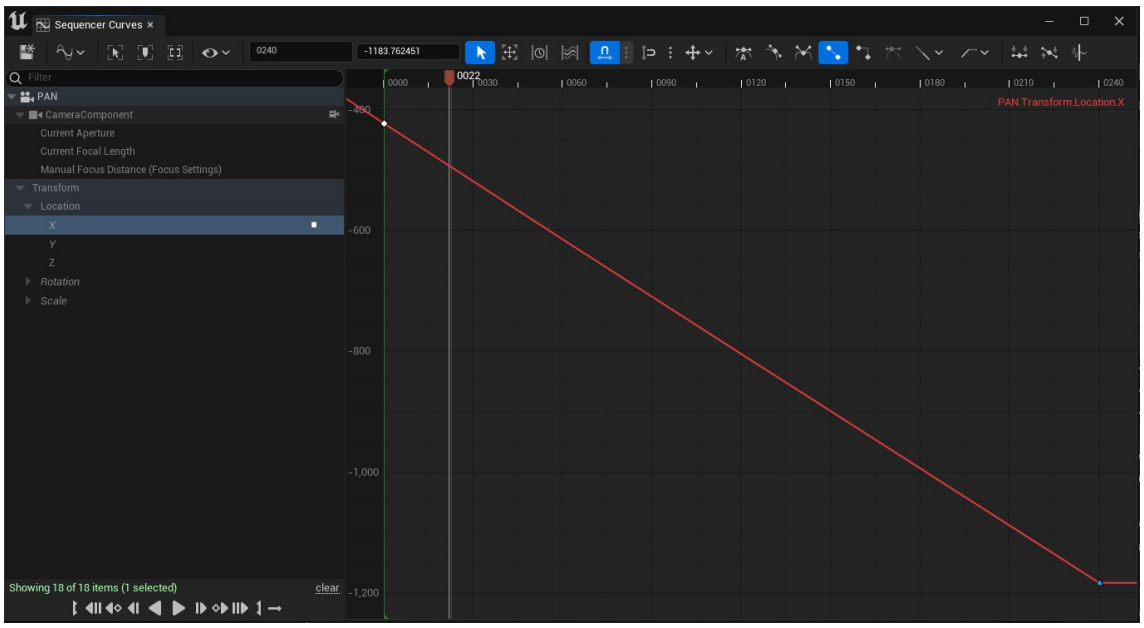

Fig. 39 Animation curves window.

After creating the camera sequence, the next step needs to link this actor reference to a new event in the level blueprint.

The blueprint will receive the event of pressing the key 3 and enabling a playblast with the stored reference of the camera sequence.

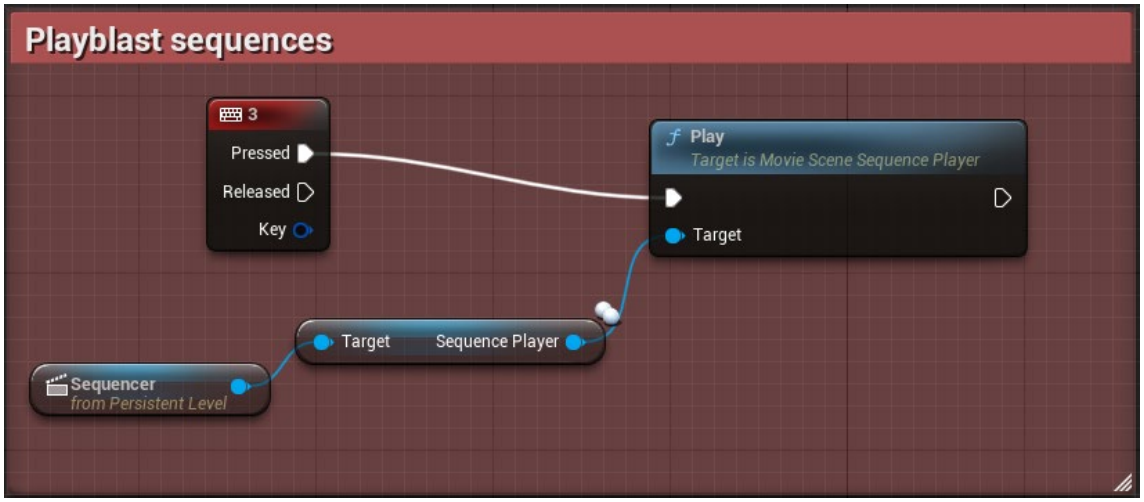

Fig. 40 3rd Blueprint. Playblast camera sequence.

5.7 Modeling

The project requires of multiple assets. During this section different modeling techniques will reach the asset that project needs.

5.7.1 NURBS Modeling

This technique has been used for a few simple and fast assets with smooth perfect surfaces such as wine glasses, vases…

In this example the modeling starts creating a Bezier curve that will define the shape of the wine glass Using Mayas revolve tool over the Bezier curve will generate the temporal shape as a NURB surface.

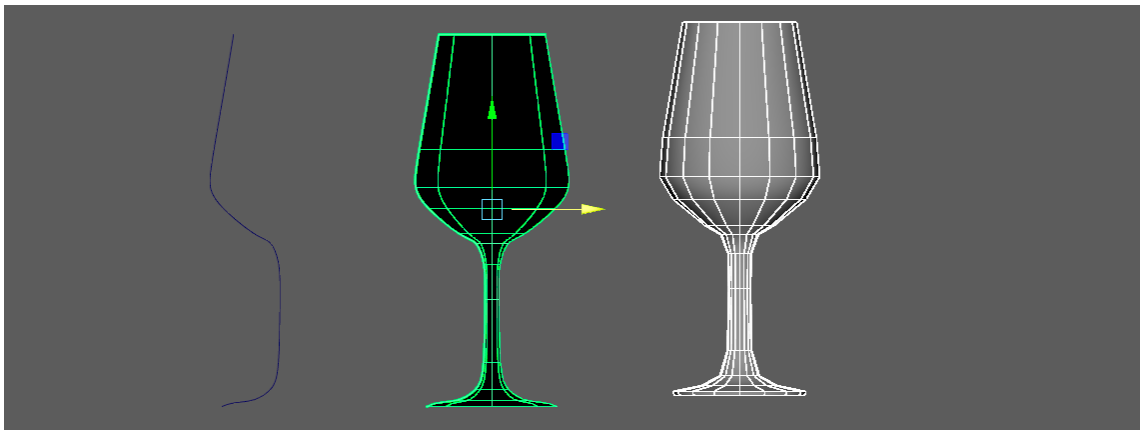

Fig. 41 Progression of NURB modeling. Curve - Surface - Polygon.

This shape's still not geometry, and it's not a 3-dimensional object that's why before exporting the asset this need to be transformed from NURB to Geometry and extruding their interior faces to generate the model.

Fig. 42 Models ready and rendered in Unreal.

Using this method, the project can create perfect surfaces related to small assets that don't require extended production time to be done.

5.7.2 Traditional Modeling

In most cases the furniture doesn't present complex geometries, many of the elements to model that can be found in the asset list require simple geometries that just need a little bit more detail in their edges for example desks, chairs legs, closets…

In the next example during the TV modeling, simple geometry with subtle detail can be found, this detail has been made using the bevel tool from Maya selecting the desired edges in the geometries that need to be subtle.

Later in Texturing 5.8 the result of texturing this TV with low polycount can be found.

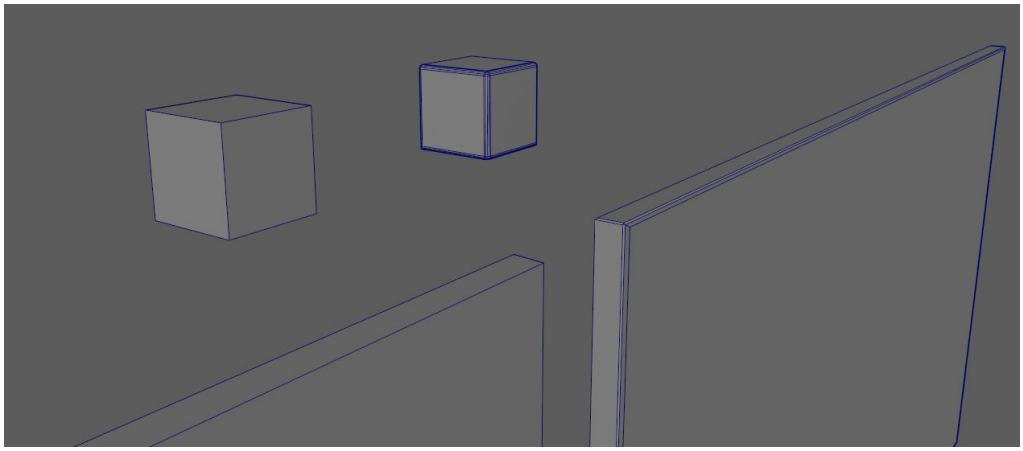

Fig. 43 Effect of beveled edges in basic shapes.

5.7.3 Subdivision

During the subdivision modeling, the asset goes through different phases. In this case, to model the WC, the first step it's placing a primitive such as a quad to start modifying that topology using the crease tool or adding edges to the geometry that gives the desired shape.

The next phase requires previewing the mesh with smooth topology using keypad 3 in Maya. Using the tools previously mentioned crease and insert edge loop the third step

will consist in transfer the shape to the model. Furthermore, moving the edges or vertex will affect the final shape too.

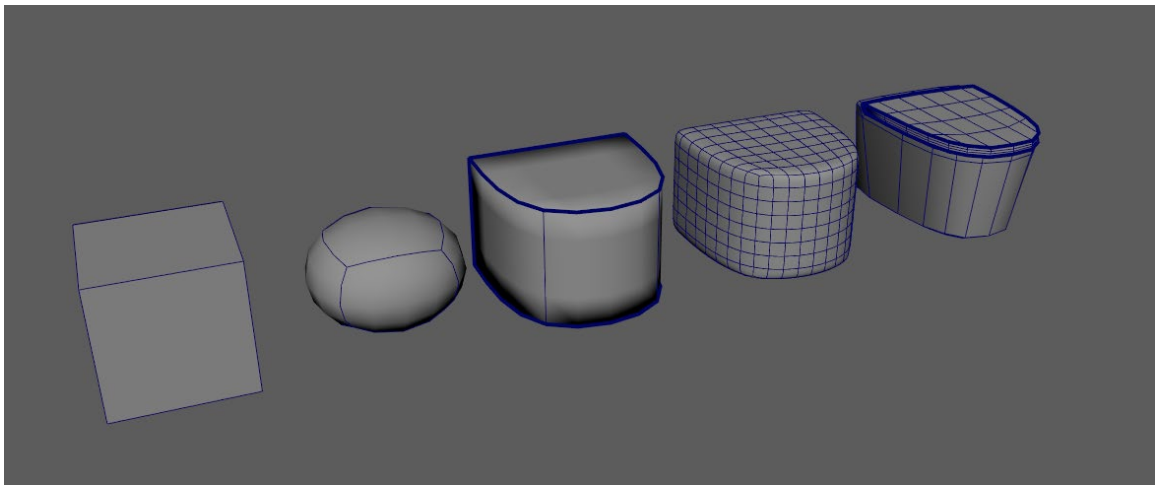

Fig. 44 Steps to reach final mesh with subdivision modeling.

When the shape is ready the asset it's still previewing how it would be with smooth geometry, that's why applying a definitive smooth with n steps is needed.

The resulting asset has many unnecessary loops that don't add volume to the geometry, which is why in the final stage the polycount is optimized and the desired final shape is given moving edges, vertex, and faces.

5.7.4 Maya nCloth Simulation

Some assets such as pillows, cushions, and bedsheets… can be done faster using Maya FX simulations related to NVidia nCloth PhysX. In this example to model a cushion, the set of steps taken was:

1. Create a primitive quad with the approximated real size
2. Applying subdivisions needed for the simulation. More the subdivisions better the result but longer simulation times.
3. Add the nCloth component to the subdivided mesh, disable gravity and play with the different simulation values like pressure, dump rate, elasticity…

4. Duplicate the desired shape from the simulation
5. Reduce polycount and store the High Poly version and Low Poly version to bake later textures.

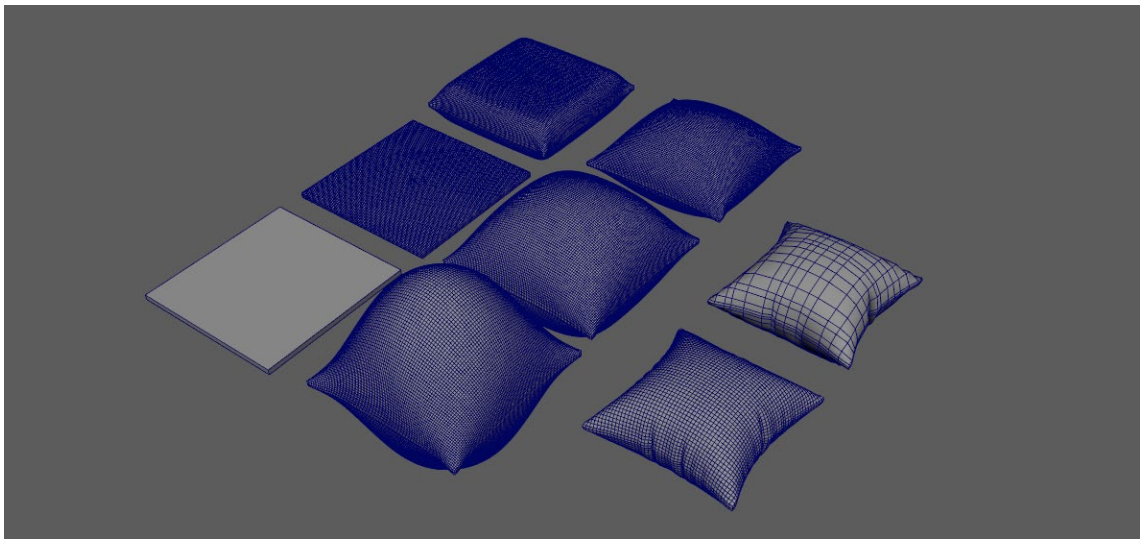

Fig. 45 Progression of modeling a cushion with nCloth simulations.

5.7.5 Boolean operations

Assets with concavities can take advantage of using Boolean operations. This tool must be used carefully because it will create inconsistent meshes, generating nGons.

Models of sinks, closets, wardrobes… as this example can use different geometries to define the final shape using union, difference, and intersection operations.

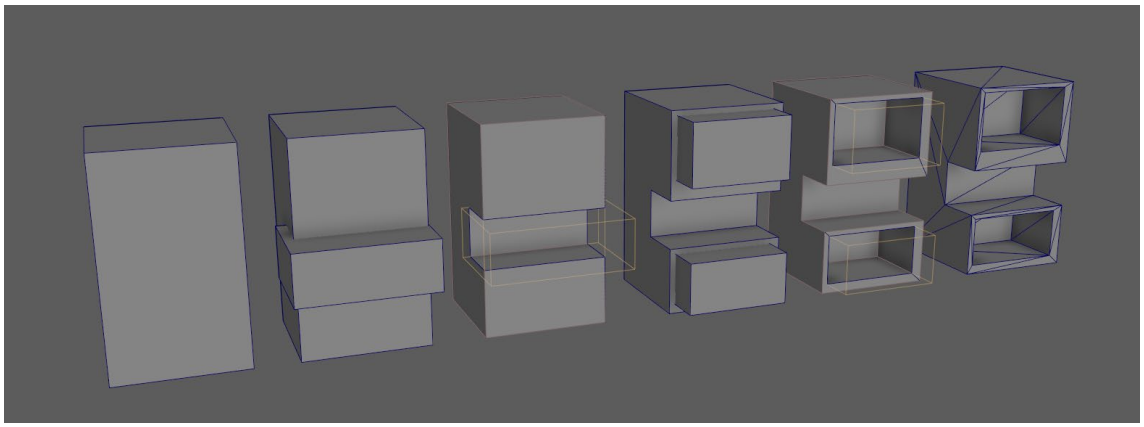

Fig. 46 Using boolean operations to reach the final result.

As can be seen in the previous image, using simple primitive geometries as quads, and modifying sizes, we can achieve different iterations of the model. The boolean tool requires selecting 2 meshes and applying the operation needed.

Finally, the model needs to be cleaned because after using Boolean operations in most cases some polygons will have more than 4-sided faces which is a bad practice for modeling even are inorganic models.

5.8 Texturing

Following the 3D pipeline, now assets that need it, must be texturized. Once the model it's ready with the set of uv's done, this needs to be exported with .obj or .fbx format to be read by Substance Painter.

Before importing the file, Substance Painter needs to set from the beginning the output texture size that will have the model. Despite Substance Painter ask at the beginning, this texture size can be modified later to larger or smaller resolution sizes using the texture set settings panel.

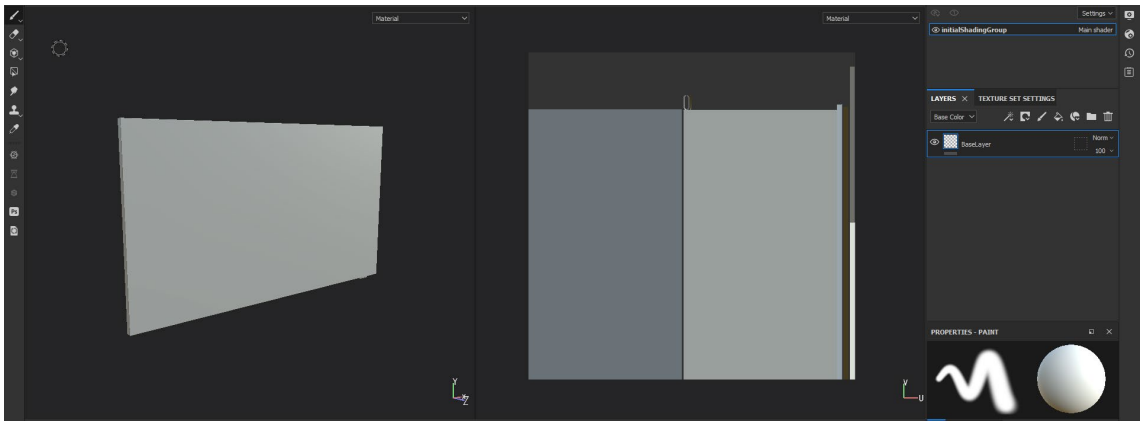

Fig. 47 Importing mesh into Substance Painter.

Once the model is loaded to the viewport, the next step requires baking the model to get a set of textures that will contain model information. Most times baking it's used to transfer information from a high-poly mesh to a low-poly preserving the detail, but in some cases, this bake can be done from low-poly meshes just to transfer information related with edges, concavities, curvature, id etc.

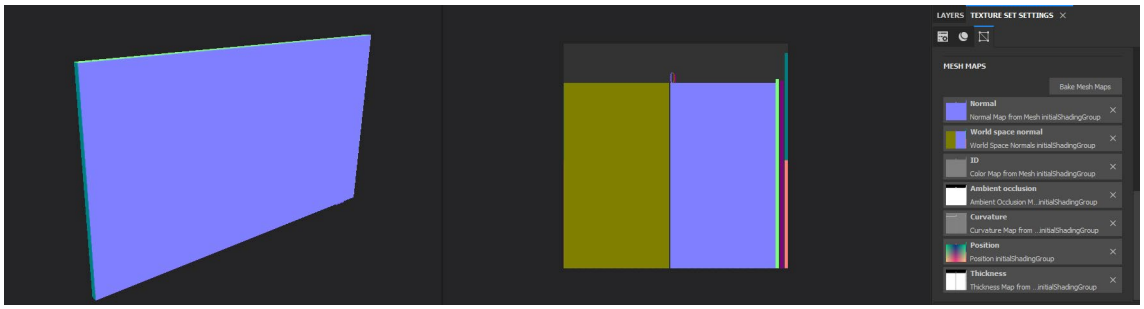

Fig. 48 Set of textures after baking the geometries.

This information is useful whenever smart materials, generators and smart mask are used. E.g., the next image contains 2 layers. The 1st layer contains a black fill that covers the mesh, the 2nd layer contains other fill in this case red.

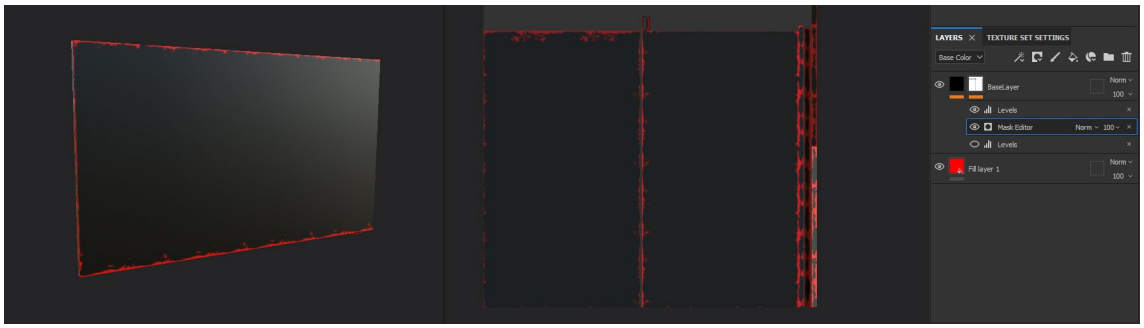

Fig. 49 Effect of using smart materials/masks into meshes with baked maps.

If we apply a smart mask over the black fill coating the red color emulating dust, scratched surfaces, or edges we can obtain similar results as seen in the figure. That's the reason why bakes even if only we have the low-poly version are useful.

Despite most of the models for this project needs to look kind of brand new and not dusty/rough/old, keeping this information it's a must-have for some textures in any other ways e.g., Cushions from high-poly version can sew the edges with this information.

This procedure has been made for each texturized asset in Substance. However, each asset has been textured in different ways.

5.8.1 Painting

The simpler technique to texture an asset requires just painting over the mesh into a layer. Different brush configurations can be applied to get the desired result.

In the following example, stitches have been created painting over the mesh(left) or UV set(right). The brush has been configured to use a stitch preset given by default in substance. Brush can be modified to set different sizes, spacing, angle, stroke opacity…

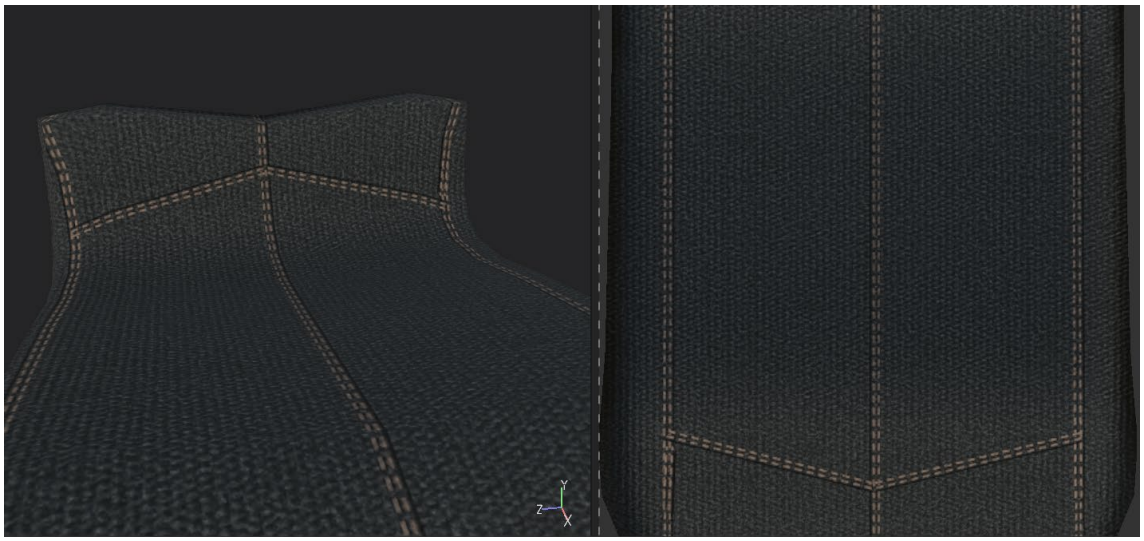

Fig. 50 Handpainting sews into the texture.

In addition, to achieve the straight stitches substance as Adobe Photoshop etc contain a tool to draw straight lines over a canvas.

5.8.2 Patterns

Some textures require the use of patterns. To use patterns a mask containing the pattern generator tool must be created into a painting layer.

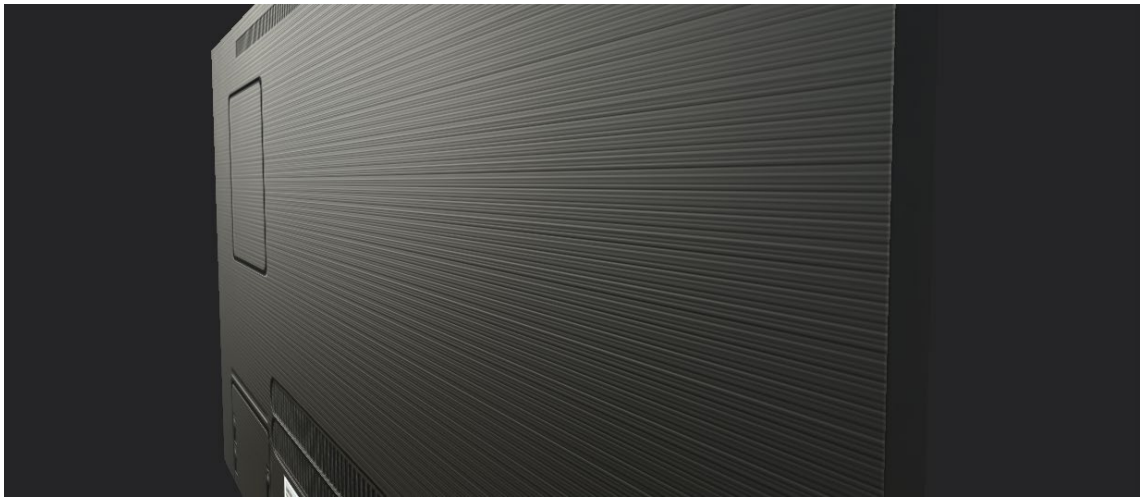

Fig. 51 Line pattern combined with height map information.

This tool can replicate multiple type of shapes (lines, squares, circles, hexagons…), allowing size, quantity, angle, and offset settings in the pattern shape. Furthermore, the tool can take advantage of the material channels adding depth into the pattern using height values or using metallic / roughness values.

5.8.3 Masking

A quick way to texture is using masks. As seen before in Patterns 5.8.2 masks can be used to split different layers into the same UV set.

In the following example, masks have been used to split the nightstand body from the drawer. To create a mask, this needs to be done into a painting layer as Photoshop.

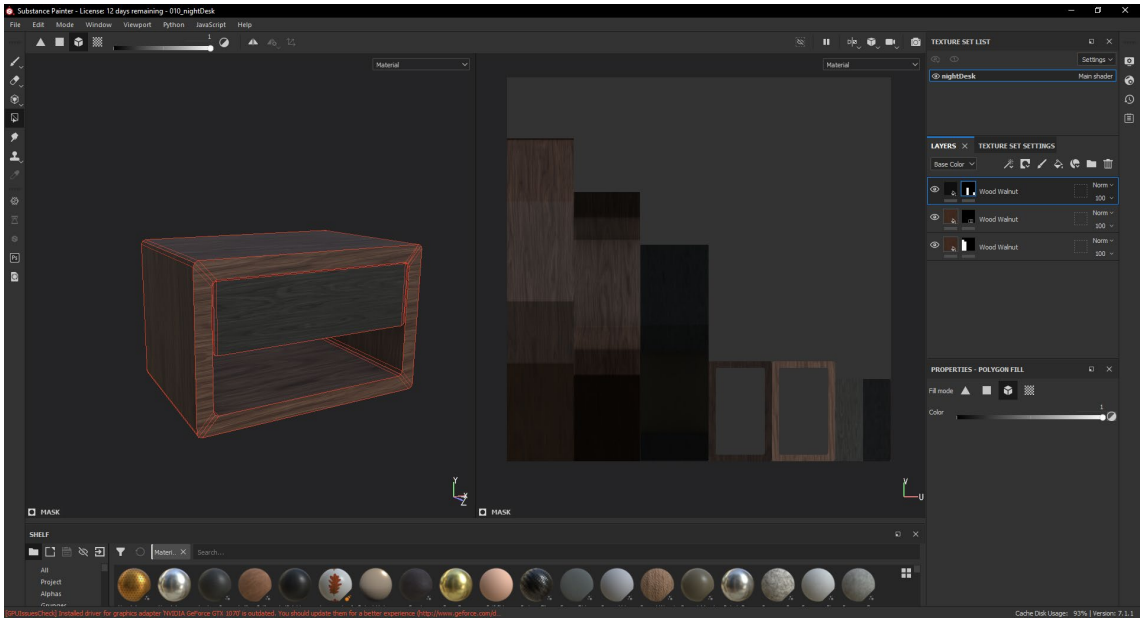

Fig. 52 Nightstand with different masks applied.

This mask can be modified to apply filters as seen at the beginning of this section or selecting specific meshes/faces/uv sets to be affected by the painting layer.

5.8.4 Fill layers

Filling layers are useful to color and giving PBR properties faster to some type of assets.

In this example using filling layers and mixing masking technique into the television the asset can obtain a reflective surface in one concrete area, in this case just the screen.

Fig. 53 Television using mask and filling layer in the screen.

As can be seen in the previous image, just the screen contains a unique filling layer. This layer owns a different color and roughness properties to replicate the reflective effect different from the tv body (corners and back).

5.8.5 Projection

This technique allows to texture a geometry using a base image. This image, it's projected into the substance viewport.

For this example, an electronic label has been taken as an image to project into the mesh. Using the label as reference and the painting tool over the projection with some

values changed in roughness and height, the television can replicate the label into the television.

Fig. 54 Projecting texture over the mesh/uvs to paint.

5.9 Building the Scene

The next step in the process requires creating the Unreal Engine scene and add the different assets done. For this purpose, the functional prototype created before will be helpful to work faster and just add the new assets every time these have been finalized into the project.

Each asset needs to be imported to Unreal using the content browser that can be seen in the following figure.

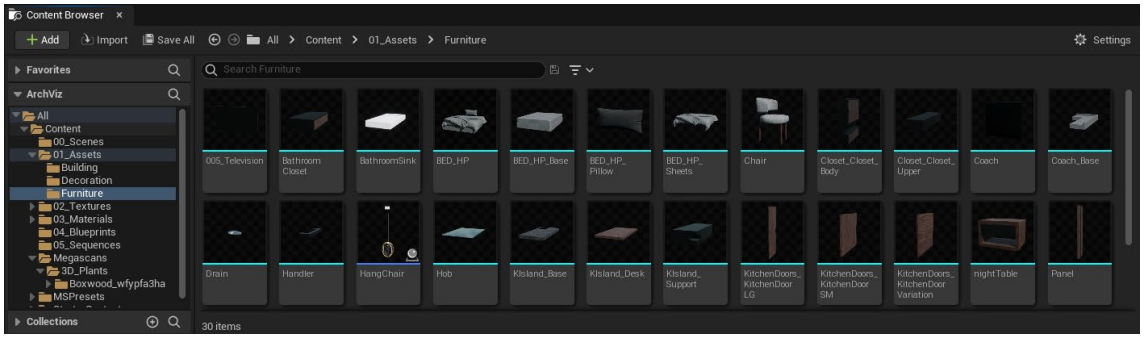

Fig. 55 Unreal Engine Content Browser.

This content browser contains every asset used in the project (meshes, textures, materials, blueprints, scenes…), that's why sorting each asset into proper content folders it's a must task when importing to track and get a fluid workflow during the project.

In addition, for each textured asset a specific task related with substance textures needs to be done. Substance generates packed texture files ready to being used in Unreal Engine. This set of textures contains diffuse, normal and a packed file with metallic, ambient Occlusion and roughness information.

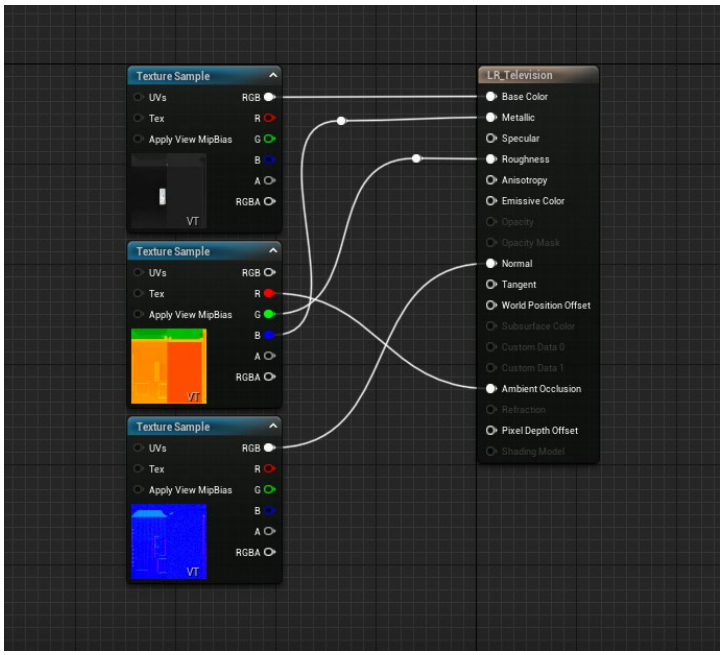

Fig. 56 Substance texture sets into Unreal Material using nodes.

In the previous figure 3 texture sets can be seen and needs to be settled into the material through nodes. The packed texture file with multiple information needs to be split through the 3 different channels: red for ambient Occlusion, blue for Metallic and green for roughness.

Building up this scene also needs a first basic lighting using common components such as directional, spot, point and rectangular lights. This basic lighting, it's helpful to get instant visual feedback about each textured asset placed into the scene.

Fig. 57 Prototype with some finished assets using basic lighting.

5.10 Lighting

In the next step the project needs achieve a realistic result, which is why lighting plays a relevant role for it.

5.10.1 Environmental Lighting

First, the scene needs to take care of natural illumination. In the first lighting conceptions, the environment was highly brightened/burnt in some areas getting difficult visibility for a daylight scene. In addition, the exposure accentuated this bad result.

Fig. 58 First light iteration in version 0.1 with visibility issues.

After the first validation in a showcase of the project with 'Miquel Bigas Tañá' (expertise in areas like photography and illumination for ArchViz), the general lighting has been changed leaving day/night lighting modes using a common technique used in ArchViz as Blue Hour.

This lighting change result of the feedback has been motivated due to my unknowledge of the existence of the previously mentioned technique "Blue hour". Blue hour allows the scene to obtain a mixture of day and night source lights mixing cool lighting (from natural sources) and warm lighting (from artificial sources) which will generate a good composition blending blue/orange complementary colors.

Fig. 59 Blue Hour example.

The easier way to obtain that result could be taken an HDRI Sky with blue lighting and using it in the scene, but due to budget reasons the project will be using the tools and components that Unreal Engine provides to reach a similar result.

Using directional light, sky atmosphere and sky light components the scene can achieve a natural source of blue light. Using the Unreal debugging view modes, the scene can be switched between Lit and Detail Lighting mode showing just the light results.

The first step needs to place a directional light avoiding an incident angle perpendicular to the windows as can be seen in the following figure.

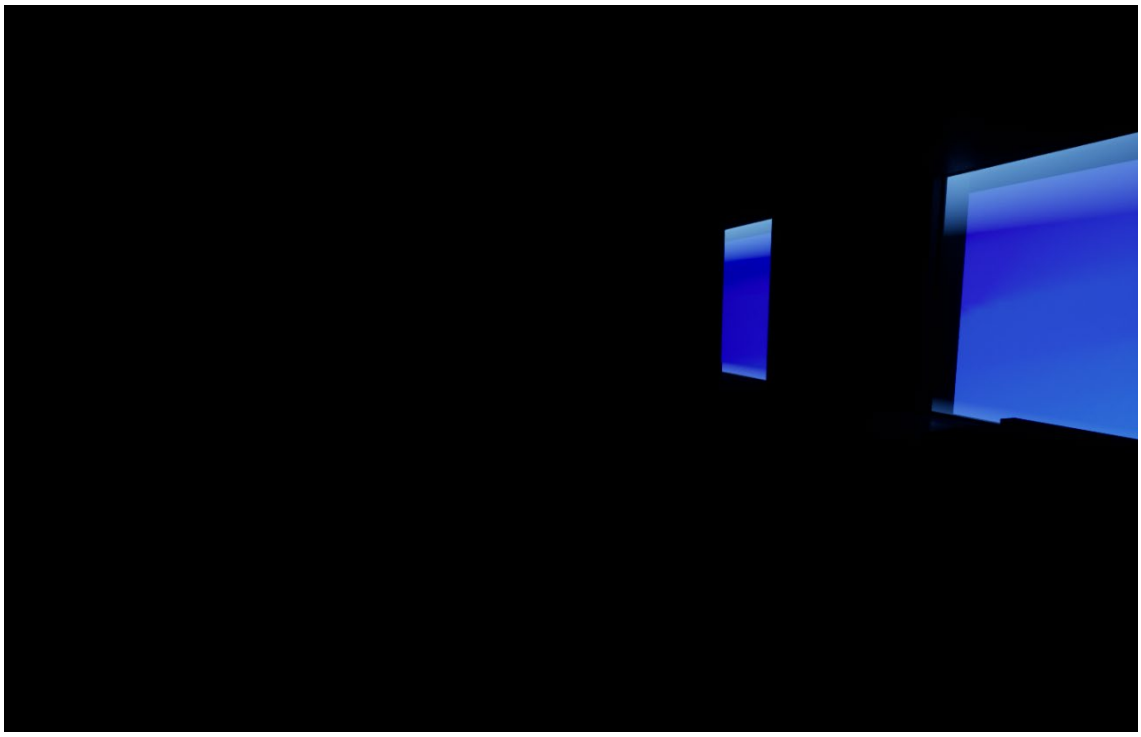

Fig. 60 First light step using directional light.

This directional light contains low-intensity values and a temperature of 9000 Kelvin to reach a cool lighting color.

In the next step using the *SkyLight* and *SkyAtmosphere* components, this directional light takes a relevant role in acting in light bounces with the components mentioned before. Moreover, *the SkyLight* component contains its own intensity and global illumination values where the general lighting of the scene will be controlled.

Fig. 61 Result of using directional light, skyAtmosphere and skyLight.

As seen in the previous figure the first results of Lumen and Global illumination can be seen in the floor and ceil.

5.10.2 Artificial Lighting and blueprint

Once the natural lighting reaches a blue hour color is done, the next step needs to replicate the artificial lighting.

Multiple area lights have been used for each section of the house (Bedroom, Bathroom, Living Room…) in addition to a couple of single spot and point lights that can be found in the bedroom. Playing with the intensity values and using a 3500 Kelvin color temperature for each light, we reach warm colors that light all the main areas.

Fig. 62 Artificial warm lighting.

Besides the artificial lighting, the scene also needed emissive light sources, creating a material that can be applied to different meshes to generate another light source (such as the lights in the ceil) affecting the global illumination of the scene.

Fig. 63 Emissive lights and global illumination.

Finally, each light element has been stored in an array, as done with cameras in the Functional Prototype Blueprint 5.6. The purpose of this array, it's to generate an event that turns on/off lighting during the app execution enabling/disabling all the elements stored in the array.

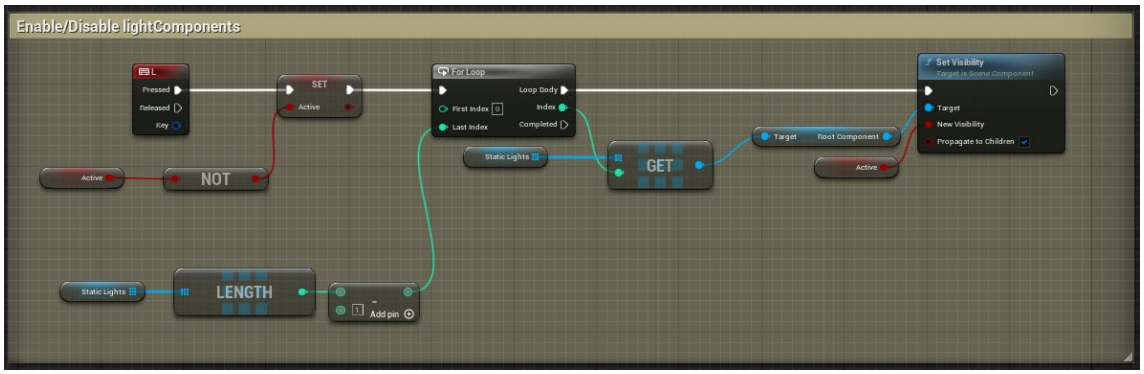

Fig. 64 Blueprint to enable/disable light sources.

5.10.3 Height Fog and Volumetric Clouds

Using height fog components and volumetric clouds will increase the realism of the scene, allowing the addition of visual elements that will improve the quality even though the scene doesn't show any environment like buildings, etc.

In the following image can be shown the difference between enabling/disabling the Volumetric Cloud component.

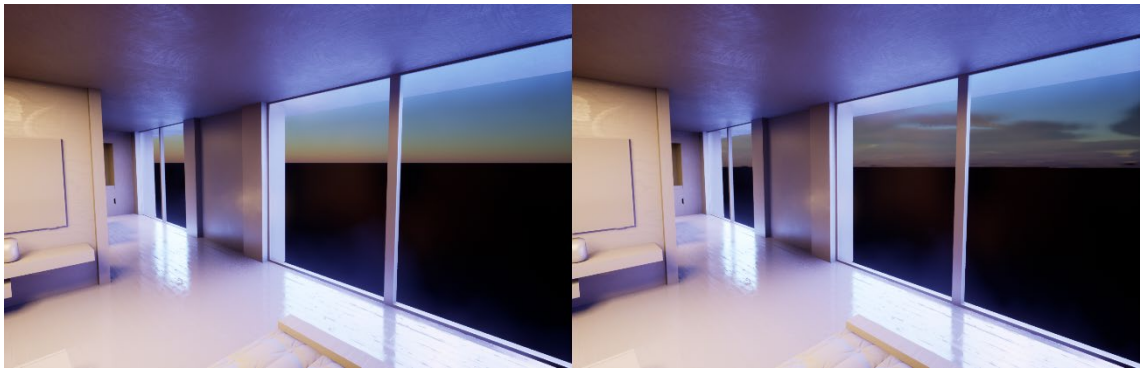

Fig. 65 Volumetric Cloud Disabled and Enabled.

In addition, using the height fog component the scene can replace the horizon black clipping plane using fog to fill the environment.

Fig. 66 Height Fog Disabled and Enabled.

Moreover, this component can be useful to add subtle volume particles as seen in State of Art 2.0-Volumetric Light into the scene using the Volumetric Fog option and increasing the fog density in the scene.

Fig. 67 High fog density value to notice the effect.

5.10.4 Lighting Result

The result of mixing both types of illumination can be seen in the following figure, where cool and warm colors blends, creating a final great-looking lighting composition.

Fig. 68 Final lighting using Blue Hour.

5.11 Post-Processing

The next step requires the use of post-processing profiles to apply effects like bloom, vignette, or depth of field into the final images.

Applying post-processing effects at the end of each render it's expensive for the GPU in relation to frame rates, therefore needs to be used carefully and that's the reason why the scene contains two profiles taking advantage of the different camera interactions.

- Profile for static cameras where the user won't notice about lower framerates.
- Lower budget profile for dynamic free camera and camera sequences where the user can notice about lower framerates.

By default, Unreal Engine cameras contain their post-process profiles where the developer can play with multiple values. During the development of this scene, this feature has been exploited to generate one profile that will be instanced containing the same values in different spots of the scene.

In the case of static cameras bloom, vignette, and in some cases depth of field effects have been enabled as can be seen in the following figure comparing both profiles.

Fig. 69 Post-Processing off/on.

5.12 Optimizations

As seen in Planning Deviation 3.6 Unreal ensures performance automatically by doing some optimizations, nevertheless, these optimizations need to be enabled.

5.12.1 Nanite vs Occlusion Culling

Nanite technology contains its compression method that optimizes each mesh. This optimization can be noticed using larger polycount meshes, but even though the project uses in most cases low poly meshes, a small performance improvement can be noticed using nanite instead of the traditional occlusion culling method related to framerates.

Nanite needs to be enabled each time a mesh it's imported into the content browser, as can be seen in the following image.

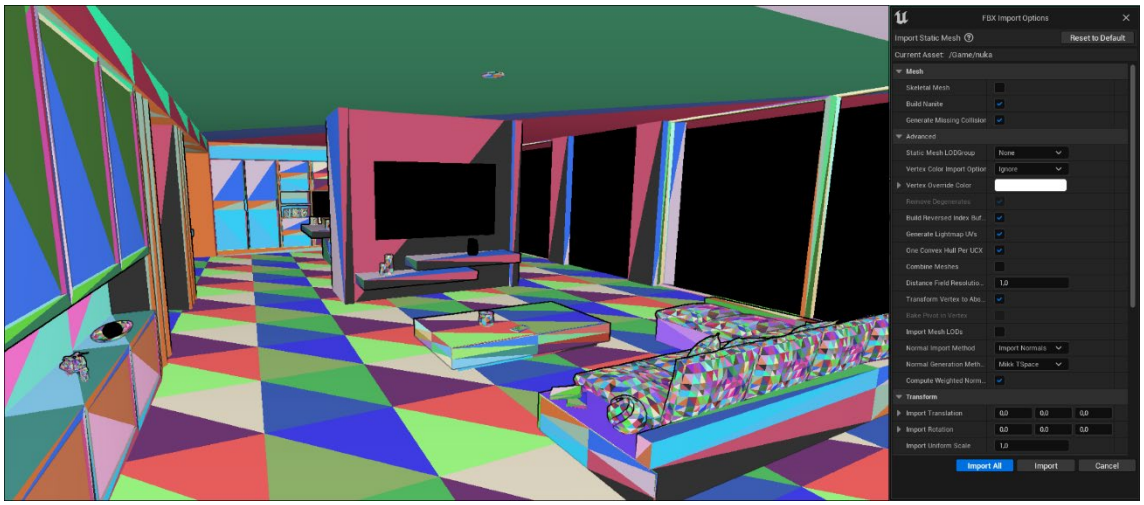

Fig. 70 Nanite clusters viewmode and importing settings window.

Otherwise, if nanite isn't enabled for any mesh this will be culled using the frustum once the camera leaves the intersection with the mesh as can be seen in the next example previously freezing a render.

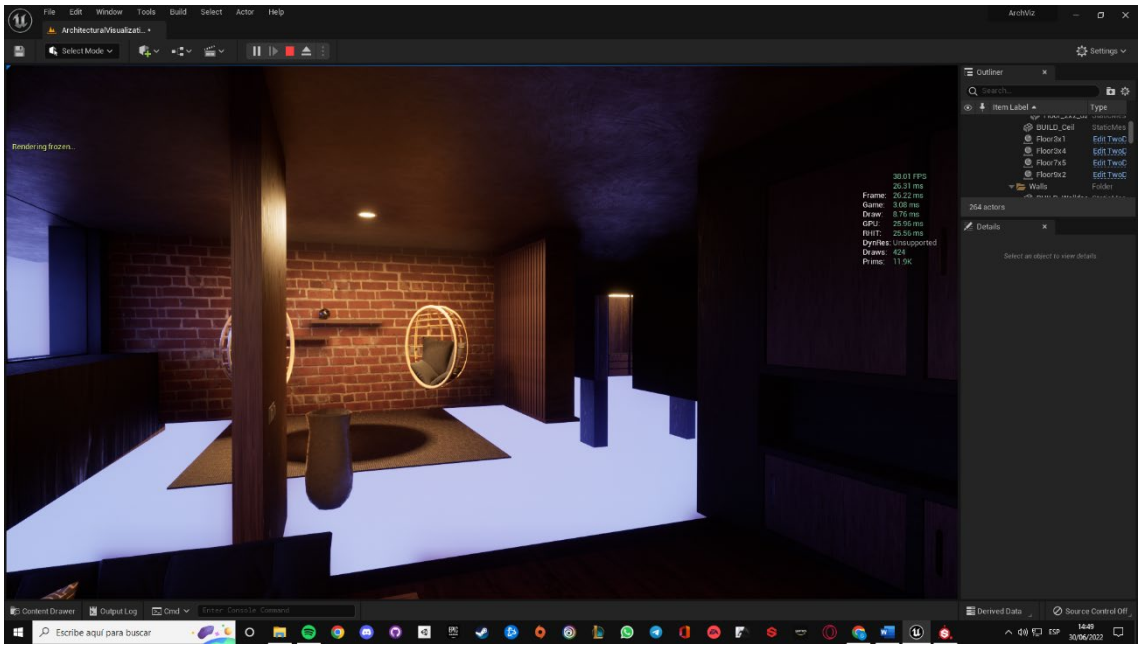

Fig. 71 Freezed render with nanite disabled in floor modules. Occlusion culling working.

In addition, Nanite can't be used with transparent materials, that's why meshes as windows that contain a transparent shader cannot be used with this technology (no triangle clusters, mesh in black).

The same example applied to foliage where alphas still generate weird rendering artifacts in the latest Unreal release.

5.12.2 Instanced rendering and Blueprints

One effective way to optimize and get faster renders implies reducing the mesh draw calls for each frame. This will increase the performance by getting higher framerates.

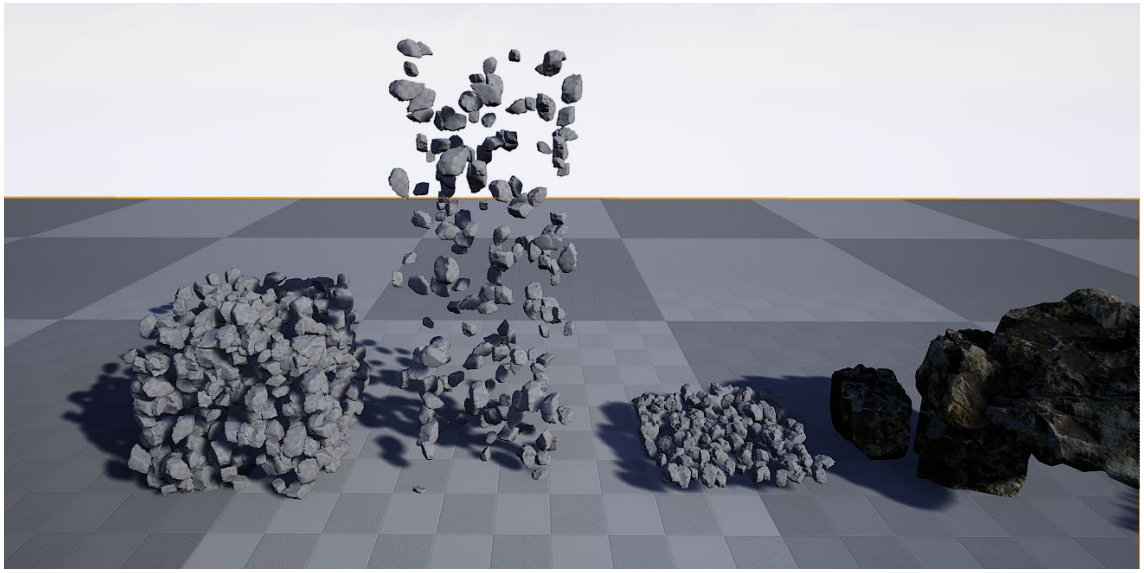

Fig. 72 Testing instances with blueprints. Rock cluster contains 2000 instanced meshes.

The project currently contains multiple repeated meshes that are not instanced and need to be rendered, such as walls, floor modules, chairs….

To reduce workforce instancing meshes another blueprint has been developed specifically to help in this task, using unidimensional and bidimensional arrays the blueprint can place instanced meshes through the scene in a few seconds.

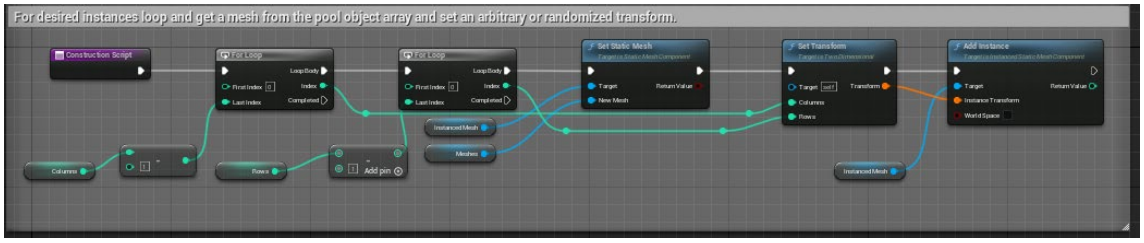

Fig. 73 Instance blueprint with bidimensional generation along 2 axes.

Using function add instance as can be seen in the previous figure will place a mesh into the desired transform position, this transforms it's given by the current array dimension-index and the offset of the mesh to be placed.

In the following image the floor is being placed using instances but many other elements such as chairs, plates, cushions, and shelves use the same principle to optimize the render queue just passing transform information to the meshes.

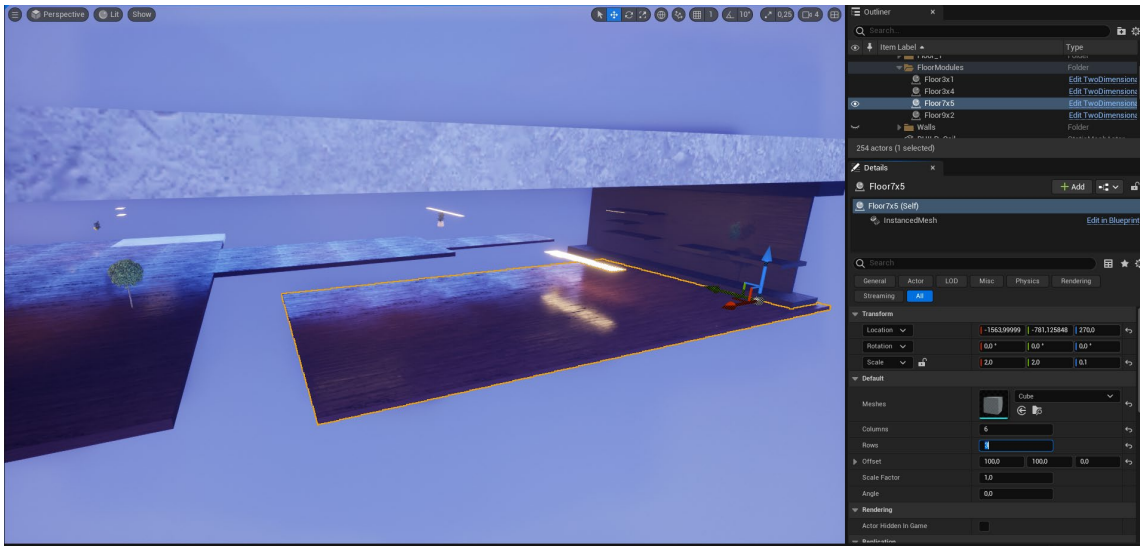

Fig. 74 Using the instance blueprint to generate floor modules.

5.12.3 LOD's

Enabled by default. Testing that LOD's are working for each mesh can be seen easily in the bedsheets (the mesh with higher polycount in the project).

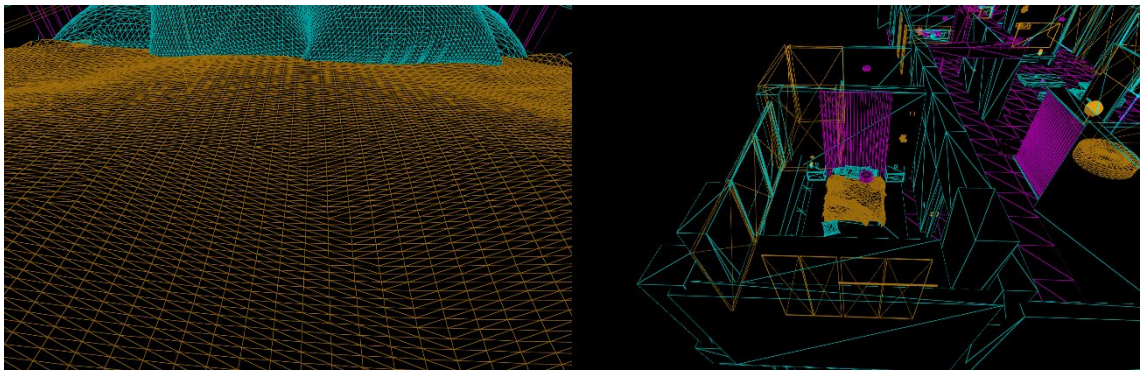

Fig. 75 Bedsheets polygon density.

In the first image the bedsheet contains a high density polycount meanwhile in the right image once the camera it's far away the geometry automatically swap between

the different LOD's showing a decrease in the polycount density in the selected mesh (orange bedsheets wireframe).

5.12.4 Asset audit

Reducing the final disk space of the project must be considered, the compilation time and the final weight of the project will be reduced for each release.

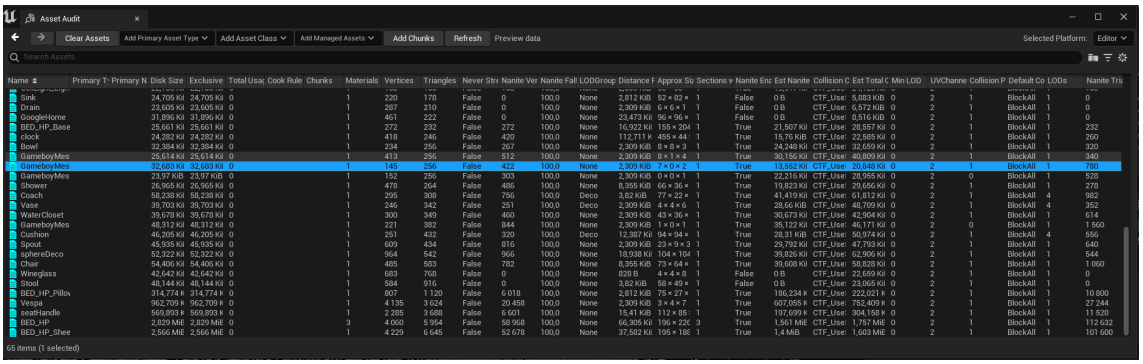

Fig. 76 Asset audit tool.

To help with this task, Unreal Engine provides a tool to Audit assets placed in the content browser. Using this tool, the project can detect which assets are not being used or bad dereferenced from memory, causing compilation exceptions.

Non-used assets such as materials, textures, scenes, and meshes… has been deleted and dereferenced properly from memory to reduce disk space. The first version of the app occupied 1,3Gb after applying asset audit, the project release in the last version reached a disk space of 657Mb.

5.13 App testing and validations

During the different stages of the project, multiple releases and tests have been done to measure the quality and performance of the final application focusing on technical aspects.

Feedback have been received in two ways:

1. Brief technical testing sessions with different devices looking for weird framerates, artifacts, and software resilience.
2. Collecting feedback from user polls after testing the app.

5.13.1 General overview

version	0.1	0.2	0.3	0.3.1	0.4	0.5	0.6	1.0
release date	13th May	19th June	20th June	20th June	21st June	22nd June	23rd June	23rd June

Table 12 Version release chronogram.

As can be seen in the previous table, the first version was released with a large time gap compared to the rest of versions that have been iterating daily with the feedback reported. This is due the first version was released during a project phase which didn't require testing and optimizations yet.

5.13.2 Changelog

Version 0.1

- Basic Illumination and Postprocess implemented.
- Blockout with some finished assets placed in scene.
- Functional prototype (static cameras, free camera, sequences...).

Version 0.2

- Polished illumination using blue lighting mixed with artificial calid lighting
- Added static viewpoints to the camera array.
- Placed finished assets in some areas.
- Escape to exit application function added.

Version 0.3

- Implemented **scalability settings** to avoid performance issues and adapt the app to different configurations.
- Quit button instead escape.
- Key bindings to display performance indicators (current framerate, GPU status...).

Version 0.3.1

- Improved and optimized UI for scalability settings.

Version 0.4

- Scene polish, more finished assets added.
- Implemented buttons to change between two resolutions *(1920x1080 and 1280x720*).
- FPS cap option now functional in settings menu.
- Fixed screen resolution by default to 1280x720px
- Enabled RTX hardware raytracing + lumen technology.

Version 0.5

- Disabled RTX hardware raytracing options due to performance and testing issues.
- Troubleshooting non-working VSync option.
- Multiple resolutions added using ComboBox instead of resolution buttons.

Version 0.6

- VSync enabled by default.
- Setting up final static camera viewpoints + postprocessing.
- Changed game settings to enable by default windowed mode.

Version 1.0

- Placed final assets.
- Static cameras adjustments, reducing post-process values as bloom effects.
- Final camera sequences and adjustments.
- Asset auditing and general memory decrease.

5.13.3 Version Troubleshooting

This section explains the resolution of the main issues detected thanks to validation and gathering information about the application after testing the app with different users and devices.

Version 0.2

The first significant release tested with other users. Caused performance issues with default values (max quality settings and 1920x1080 resolution) in GPUs without hardware raytracing.

To fix these performance issues, later in versions, 0.3 and 0.3.1 a scalability settings menu has been developed creating a whole User Interface that enables/disables the settings menu with escape key where the user can change the quality settings.

In the next image, the first version of the UI can be seen (0.3 release). This UI was inefficient and not flexible to changes due to the multiple calling events for the different settings (4 options x 8 rows = 32 different button event calls). In addition, the VSync and Framerate settings aren't functional. Finally, users reported that buttons didn't display enough visual feedback about the selected option.

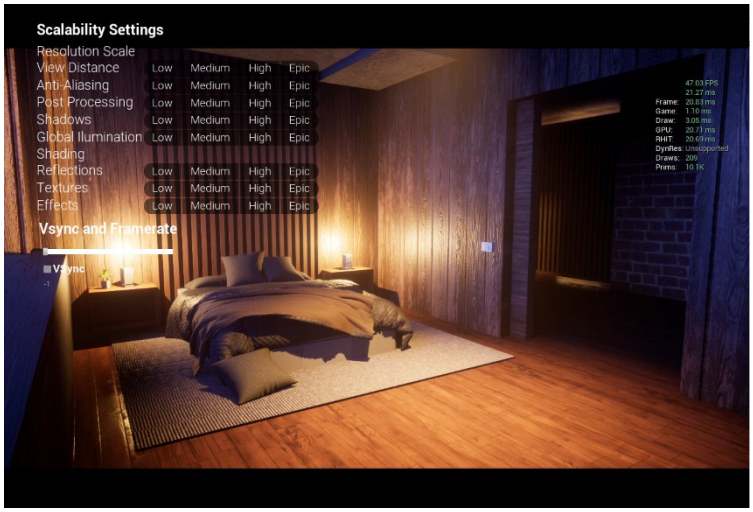

Fig. 77 First Scalability settings UI (0.3 version).

Even though UI and user-centered design it's not the main goal of this project the UI needed a new look, more efficient and flexible to work with, that's why a 0.3.1 version was released changing buttons instead of sliders that will have a stepped slide between 4 values:

- 0 Low Quality
- 1 Medium Quality
- 2 High Quality
- 3 Ultra Quality

These integer values are useful to change internally the quality settings of the project, calling a command function each time the slider has been modified.

E.g *sg.ShadowQuality 0* | *sg.ViewDistance 3* | *sg.AntialiasingQuality 2*

Fig. 78 Second Scalability Settings UI with settings at Ultra(0.3.1).

The figure above shows the new aspect of the UI Scalability Settings. Moreover, a new feature as Screen Resolution settings it's implemented but not functional. As can be

seen in the image the app can reach 46fps during the execution in static view mode using the ultra-settings at 1080x720.

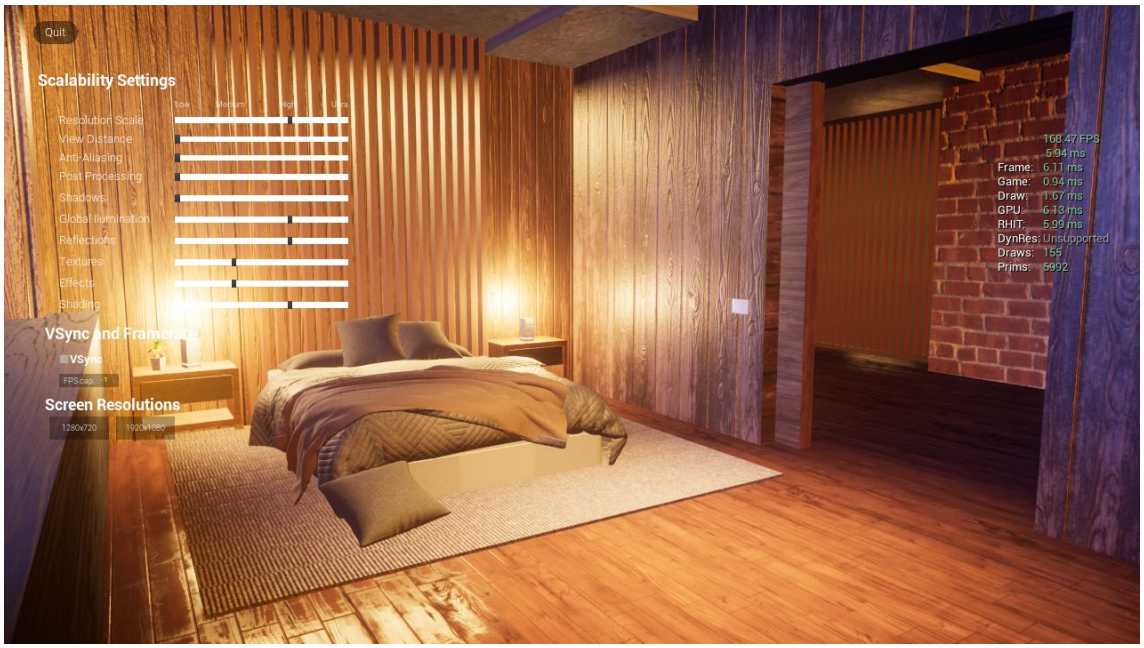

Fig. 79 Scalability setting values modified.

On the other hand, in this image, some quality setting values have been modified (no shadows, less resolution scale, and texture quality…) the app reaches a peak of 168fps which means that scalability settings are working.

Version 0.4

The next big troubleshooting can be found in this version. During this release, resolution buttons and FPS cap input is seen before became now functional.

Furthermore, as seen in 5.11 Post-Processing, each Unreal camera contains different rendering profiles.

Within these profiles, the project can increase the lumen quality settings and enable the hardware raytracing for each camera.

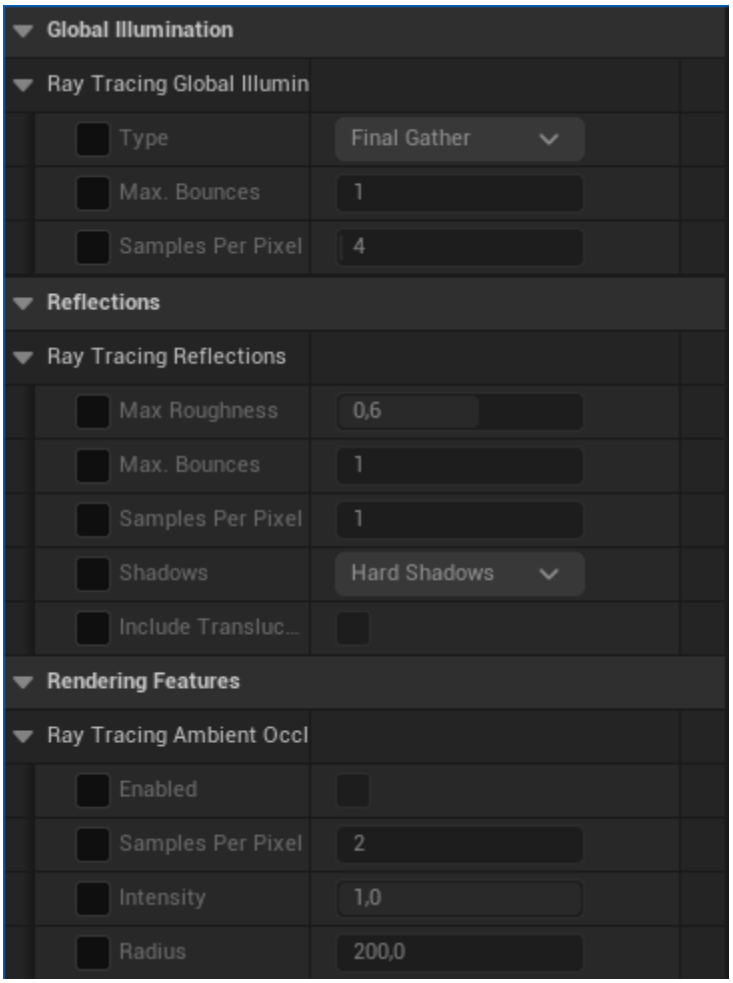

Fig. 80 Camera Profile. Raytracing parameters.

This last change caused performance issues in users who owned raytracing GPUs, getting worse framerates in the same spots as users without raytracing fact which was unusual.

This problem meant permanently deactivating the raytracing enhancement options in the 0.5 release since I was to work blindly because I don't own a raytracing graphic card that allows me to have hardware references.

Version 0.5

With the previous raytracing trouble fixed in this version now, screen resolution settings have been improved implementing a ComboBox list instead of two buttons giving the users multiple screen resolutions.

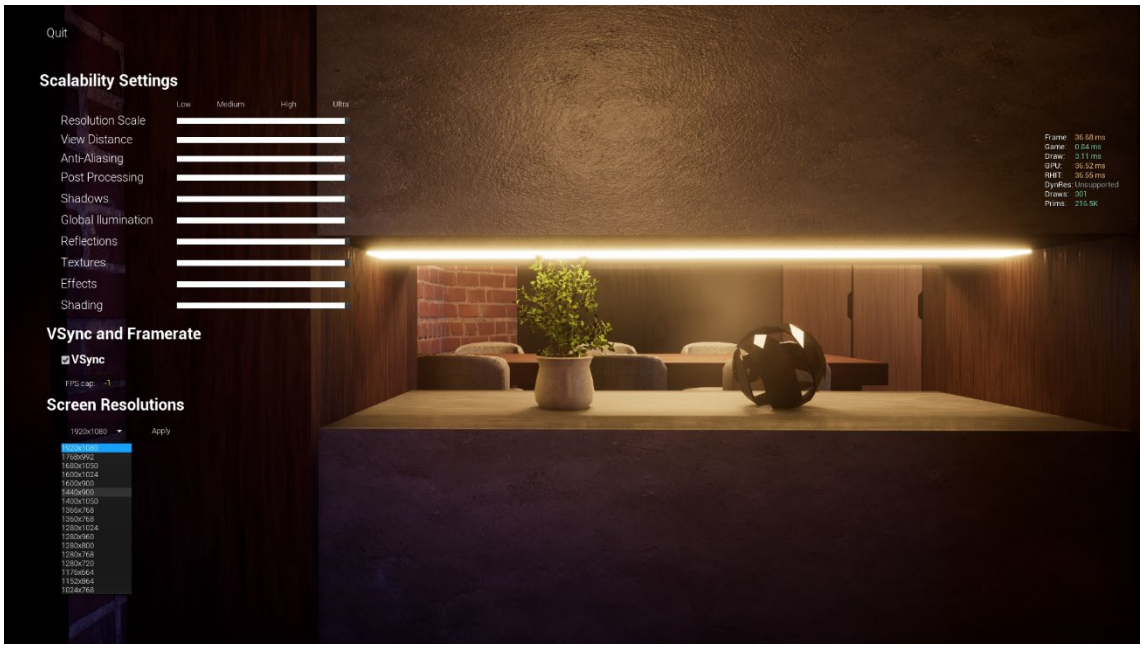

Fig. 81 Screen resolution comboBox.

This change supposed that users couldn't change resolutions due to a bad implementation forcing users to use the app on full screen without switching screen resolutions. Furthermore, the VSync option wasn't working as expected, being disabled since the beginning causing tearing problems during the camera sequences.

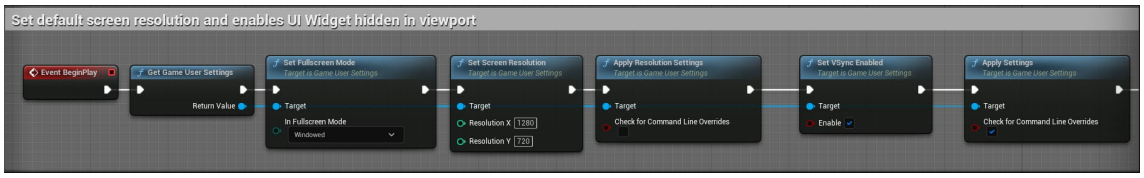

Fig. 82 Level blueprint enables by default Vsync and Windowed mode.

To solve these issues looking for a 0.6 version the level blueprint has been modified setting by default at the beginning of app execution a Windowed mode at 1280x720 resolution and enabling the VSync setting as can be seen in the previous figure.

5.13.4 Validation results

After testing the app among 7 different versions the project generated a performance stable release found in version 1.0.

This release has been validated using google forms testing with technical user profiles.

During the validation process we found some valuable information related to graphic cards:

- 30% of the users testing the app had a Raytracing dedicated GPU.
- 10% of users didn't have any dedicated GPU to test the app and used the CPU's integrated graphics card (Intel HD Graphics).

The previous information it's relevant to get some conclusions about budget and performance. By default, using a Full HD resolution with all settings at max quality users performed an average of 32 fps which it's very acceptable with these quality settings.

Some users testing the app experienced performance issues at max settings using non-raytracing supported GPUs (older than Nvidia 10 series released 27th May 2016). However, these performance issues could be fixed in 100% of the cases through the scalability and resolution settings developed for this purpose in versions 0.3 and 0.3.1.

In conclusion, testing the app we found that performance issues can be easily fixed using the settings menu. Users with non-raytracing graphic cards did experience global illumination in real-time even in the integrated CPU graphic cards with lower quality settings.

Optimizations developed throughout this project have worked and allow the application to adapt to different device configurations to reach stable performance values.

6. Conclusions

Personally, I'm pleased with the result despite my lack of knowledge in design and architecture, failure was a possibility even if we add that I've never worked with Unreal Engine as seen in SWOT Analysis 3.3.

Learning this tool from zero during the project development was a big barrier for me, but I've been able to learn about software development in Unreal during this period. Reaching the result of this project made me learn a lot of Unreal including blueprint programming, optimizations, illumination, camera configurations, animations… even though the engine learning curve it's difficult in my opinion, due to the great amount of features/tools that the engine provide and need to be learned to harness the full potential of the engine.

Unreal it's a powerful tool not only for game development but also for many other projects outside this scope as seen in Market analysis 2.4. It has an exceptional range of possibilities. Lumen technology it's a must-have in development nowadays, generating global illumination in real-time powered by graphic cards without hardware raytracing with impressive results, likewise Unreal breaks the optimization barriers in geometry with Nanite. There's still much to explore and master, greater results will be reached in a future.

To sum up, all the initial objectives for the project have been accomplished, showing during the journey a variety of modeling/texturing techniques, and going through optimizations doing some code snippets with blueprint programming.

6.1 Future applications

The project in the current state can be improved in some ways, leaving the technical fields seen along with this project. Switching the project scope to a user-centered design can improve/upgrade the project, allowing the users to:

- VR Support to navigate through the scene.
- Environmental interactions (day and night cycles, animations, sounds)
- Scene settings with a supported User Interface. E.g., setting values such as light intensity, current time, enable/disable lighting to interact in real-time with the scene and global illumination changes.

7. Bibliography

Websites

A. (2021, 30 diciembre). *What Is NURBS Modeling?* ITS. Recuperado 20 de marzo de 2022, de https://it-s.com/what-is-nurbs-modeling/

AutoDesk. (s. f.-a). *NURBS Modeling*. Maya 2020 | Autodesk Knowledge Network. Recuperado 20 de marzo de 2022, de https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Modeling/files/GUID-735A0B9A-2180-4FB8-9A7B-68F21F306E97-htm.html

AutoDesk. (s. f.-b). *Polygonal Modeling*. Maya 2020 | Autodesk Knowledge Network. Recuperado 20 de marzo de 2022, de https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2020/ENU/Maya-Modeling/files/GUID-7941F97A-36E8-47FE-95D1-71412A3B3017-htm.html

Autodesk. (s. f.). *Software Maya | Obtener precios y comprar el producto oficial Maya 2022*. Recuperado 3 de marzo de 2022, de https://www.autodesk.es/products/maya/overview?term=1-MONTH&tab=subscription

Crowder, A. (2022, 2 febrero). *3D Scanning and Photogrammetry Explained | VNTANA*. VNTANA | 3D CMS & AR eCommerce. Recuperado 21 de marzo de 2022, de https://www.vntana.com/blog/3d-scanning-and-photogrammetry-explained/

Epic Games. (s. f.). *Unreal Engine Licensing Options*. Unreal Engine. Recuperado 3 de marzo de 2022, de https://www.unrealengine.com/en-US/download

Glassdoor - Junior 3D Artist Salary. (s. f.). Glassdoor. Recuperado 19 de marzo de 2022, de https://www.glassdoor.es/Salaries/junior-3d-artist-salary-SRCH_KO0,16.htm?countryRedirect=true

Glassdoor - Junior Developer Salary. (s. f.). Glassdoor. Recuperado 19 de marzo de 2022, de https://www.glassdoor.es/Sueldos/espa%C3%B1a-junior-developer-sueldo-SRCH_IL.0,6_IN219_KO7,23.htm?clickSource=searchBtn

Global Illumination - Viz Artist and Engine. (s. f.). Vizrt. Recuperado 3 de marzo de 2022, de https://documentation.vizrt.com/viz-artist-guide/4.0/Global_Illumination.html

Global Illumination Across Industries (SIGGRAPH 2010 Course). (s. f.). SIGGRAPH. Recuperado 18 de marzo de 2022, de https://cgg.mff.cuni.cz/%7Ejaroslav/gicourse2010/

Kuzmin, V. (2019, 8 septiembre). *Full Photogrammetry Guide for 3D Artists*. 80lvl. Recuperado 21 de marzo de 2022, de https://80.lv/articles/full-photogrammetry-guide-for-3d-artists

LearnOpenGL - Frustum Culling. (s. f.). LearnOpenGL. Recuperado 3 de marzo de 2022, de https://learnopengl.com/Guest-Articles/2021/Scene/Frustum-Culling

LearnOpenGL - Textures. (s. f.). LearnOpenGL. Recuperado 4 de marzo de 2022, de https://learnopengl.com/Getting-started/Textures

Pixologic. (s. f.). *ZBrush 2022*. ZBrush. Recuperado 3 de marzo de 2022, de https://store.pixologic.com/zbrush-2022/

Publishing, P. (2020, 29 abril). *How Does Photogrammetry Work?* PhotoModeler. Recuperado 21 de marzo de 2022, de https://www.photomodeler.com/how-does-photogrammetry-work/

Epic Games. (s. f.-a). *Nanite Virtualized Geometry*. Nanite Virtualized Geometry. Recuperado 7 de abril de 2022, de https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/

Epic Games. (s. f.-b). *Unreal Engine 5.0 Release Notes*. Unreal Engine Documentation. Recuperado 6 de abril de 2022, de https://docs.unrealengine.com/5.0/en-US/unreal-engine-5-0-release-notes/

Karis, B., Stubbe, R., & Wihlidal, G. (s. f.). *Nanite SIGGRAPH Advances 2021*. Nanite SIGGRAPH. Recuperado 7 de abril de 2022, de https://advances.realtimerendering.com/s2021/Karis_Nanite_SIGGRAPH_Advances_2021_final.pdf

Lumen Global Illumination and Reflections. (s. f.). Lumen Global Illumination. Recuperado 7 de abril de 2022, de https://docs.unrealengine.com/5.0/en-US/lumen-global-illumination-and-reflections-in-unreal-engine/

McKenzie, T. (2021, 16 agosto). *A Deep Dive into Unreal Engine's 5 Nanite*. Nanite Technology. Recuperado 12 de mayo de 2022, de https://80.lv/articles/a-deep-dive-into-unreal-engine-s-5-nanite/