Escuela de Doctorado
y Estudios de Posgrado
Universidad de La Laguna

MÁSTER UNIVERSITARIO EN DESARROLLO DE VIDEOJUEGOS

Trabajo Fin de Máster

Rutas patrimoniales en La Laguna, ambientadas en el siglo XVI, para dispositivos de Realidad Virtual. Actualización y optimización de personajes.

Heritage routes in La Laguna, set in the 16th century, for Virtual Reality devices. Character update and optimization.

Autor: Juan Siverio Rojas

D./Dña. **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora Contratada Doctora adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutora.

D./Dña. **Fernando Andrés Pérez Nava**, con N.I.F. 42.091.420-V profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

“Rutas patrimoniales en La Laguna, ambientadas en el siglo XVI, para dispositivos de Realidad Virtual. Actualización y optimización de personajes” ha sido realizada bajo su dirección por D. Juan Siverio Rojas, con N.I.F. 78.556.096-X

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en Lugar a Fecha.

Agradecimientos

A mi familia, especialmente a mi esposa

A Manuel González Mauricio

A Isabel Sánchez Berriel

A Fernando Andrés Pérez Nava

A todo el profesorado del Master oficial en Desarrollo de Videojuegos de la ULL 2022/2023

Licencia

Resumen

Este proyecto es la implementación de una aplicación para realidad virtual utilizando el motor de videojuegos Unity, en su versión 2023, con el objetivo de simular el aspecto de la ciudad de San Cristóbal de La Laguna del siglo XVI mediante una reconstrucción arquitectónica basada en los mapas del arquitecto de la época Leonardo Torriani.

Aunque el trabajo se realiza comenzando un proyecto nuevo desde cero, se dispone de multitud de material disponible, como casas, muros, terreno, árboles, personajes y complementos. Además, en lo referente a la programación, este proyecto es la evolución de otros ya maduros y en estado de desarrollo avanzado, aprovechando así, técnicas de optimización, ideas y *gameplay* ya utilizadas.

Palabras clave: Realidad Virtual, HMD, Torriani, La Laguna, optimización, Lod, ShapeKeys, Reconstrucción.

Abstract

This project is the implementation of a virtual reality application using the Unity video game engine, in its 2023 version, with the target of simulating the appearence of the 16th-century city of San Cristóbal de La Laguna through an architectural reconstruction based on the maps of the architect Leonardo Torriani from that era. Although the work is started from scratch, there is a wealth of available material, such as houses, walls, terrain, trees, characters, and accessories. Furthermore, in terms of programming, this project builds upon previous mature projects in an advanced stage of development, thus benefiting from optimization techniques, ideas, and gameplay already employed.

Keywords: Virtual Reality, Torriani, La Laguna, Optimization, Lod, ShapeKeys, Reconstruction

Contenido

Índice de Ilustraciones

Índice de Tablas

Capítulo 1
Introducción

La realidad virtual es bien conocida por todos, pero en la actualidad, la mejora de las prestaciones y el acceso a los dispositivos necesarios a un precio asequible, han hecho que sea factible su utilización por el usuario doméstico, ampliando enormemente la cantidad de público final que puede tener acceso y disfrutar de esta tecnología. Incluso, la programación para estos dispositivos es ahora posible no solo para grandes estudios, sino también para desarrolladores independientes.

Esto abre un gran abanico de posibilidades y en este caso, la aplicación de Realidad Virtual que se expone en este trabajo es la de representar La Laguna del siglo XVI, basada en el mapa del arquitecto, ingeniero y cartógrafo italiano y militar Leonardo Torriani.

Con esta aplicación, se puede vivir una experiencia bastante inmersiva de lo que sería visitar y pasear por los lugares más emblemáticos de la ciudad de La Laguna en esa época. Las edificaciones, monumentos históricos, vestimenta y por supuesto ambientación, intentan ser lo más fieles posibles a lo que se viviría en la época del siglo XVI.

1.1 Antecedentes y estado actual.

Este proyecto lleva ya varias versiones previas de desarrollo, pero este trabajo se centra en las versiones creadas para realidad virtual, en la que ya hay también varias ediciones [1],[2],[3]. Actualmente, la recreación de escenarios en cuanto a cantidad de personajes y otras características gráficas, es bastante reducida, debido fundamentalmente a la alta calidad en cuanto a texturas y número de vértices de las mallas creadas por los diseñadores y a las

limitaciones hardware de los dispositivos para realidad virtual (HMD, por su nombre en inglés), funcionando en modo *standalone* (sin estar conectadas a un PC). Aunque se ha mejorado bastante con optimizaciones realizadas en trabajos anteriores, sería necesario aumentar aún más el rendimiento para mejorar así la fluidez, al menos en versiones *standalone* de realidad virtual.

En el estado actual del proyecto "Reconstrucción Histórica de San Cristóbal de La Laguna", cuya versión más actual data de septiembre de 2022 [3], nos encontramos como punto de partida con las siguientes características:

- Proyecto funcional para las Meta Oculus Quest 2.
- Escenario de edificios singulares, casas, terrenos y variedad de personajes.
- Desarrollado con el motor de videojuegos Unity en su versión 20.19.4.9.f1
- Vertex Animations Tools (VAT)[10]: Herramienta de pago que permite realizar animaciones basadas en vértices.

***Ilustración 1**. Catálogo de personajes*

1.2 Objetivo Inicial

El objetivo de este trabajo será partiendo de la última versión del proyecto, realizar su conversión hacia el motor de Unreal, buscando sobre todo una optimización del rendimiento, puesto que especialmente en lo que se refiere a los personajes, se muestra una funcionalidad limitada con las Meta Oculus Quest 2 en modo standalone, debido al gran consumo de recursos que se necesita.

El principal motivo es la cantidad de vértices de que componen la geometría de las mallas de los personajes y terreno, habiendo resultado insuficientes, para un desarrollo fluido, las optimizaciones realizadas en anteriores proyectos.

1.3 Análisis del proyecto y profiling

Tras un análisis exhaustivo del material que se dispone, se encuentran muchas partes susceptibles de ser optimizadas, y que dan a entender que la conversión a Unreal, objetivo principal, en caso de que resultara en una mejora del rendimiento, realmente no sería algo notable.

Así que aún a riesgo de no conseguir finalmente resultados óptimos, se decide que merece la pena investigar posibles alternativas, dejando como objetivo secundario la conversión a Unreal.

El ordenador de desarrollo utilizado tiene las siguientes prestaciones:

- Intel Core I5-4460 3.2 Ghz
- 16 GB RAM DDR3
- Nvidia Geforce RTX 3060
- Unidad de almacenamiento SSD Samsung Evo 870 1TB.

Con el hardware descrito, y tras un análisis del funcionamiento de la aplicación y profiling de la misma, se considera que los puntos susceptibles de ser mejorados, con los que se cree sería más notable un aumento del rendimiento, son los siguientes:

- **Tamaño:** El tamaño del proyecto sin compilar es de 15 Gb, lo cual se traduce en que apenas se puede trabajar con éste sin experimentar una lentitud intratable, produciéndose incluso bloqueos y cuelgues esporádicos del editor de Unity.

- **LOD.** El *"Level of detail"*, parece ser una técnica prometedora que no está implementada. Consiste en renderizar diferentes calidades de mallas dependiendo de la distancia a la que se encuentre el personaje respecto a la cámara, permitiendo así bajar la calidad del renderizado en los objetos que se encuentran más alejados.

- **Occlusion Culling:** Con respecto al escenario, se aplica una técnica llamada Oclusión Culling, la cual consiste en dejar de renderizar aquellos objetos que no son visibles desde el punto de vista de la cámara, ahorrando así recursos. Debido a que esta técnica solo funciona a nivel de malla completa, o sea, no se puede ocultar de forma parcial un objeto, se realizó una optimización en trabajos anteriores que consistió en dividir el terreno en triángulos, permitiendo así, ocultar aquellas partes de terreno que no se muestran en un momento dado.

  Sin embargo, esto generó una malla por cada triángulo, provocando como contrapartida, casi 6,000 GameObjects (objeto básico en una escena de Unity) solo para el terreno, con el consiguiente consumo de memoria y recursos.

- **VAT.** La animación de los personajes se realiza mediante un plugin de pago llamado Vertex Animation Tools VAT (por sus siglas en inglés), el cual permite realizar animaciones por

vértices mediante un formato denominado *point caché*. Los *assets* generados para cada personaje por esta aplicación, son extremadamente pesados. En la imagen inferior se puede ver que, en el caso del personaje de una mujer humilde, solo la animación en la que está hablando junto con la animación de caminar, ya suman casi 500 Megabytes.

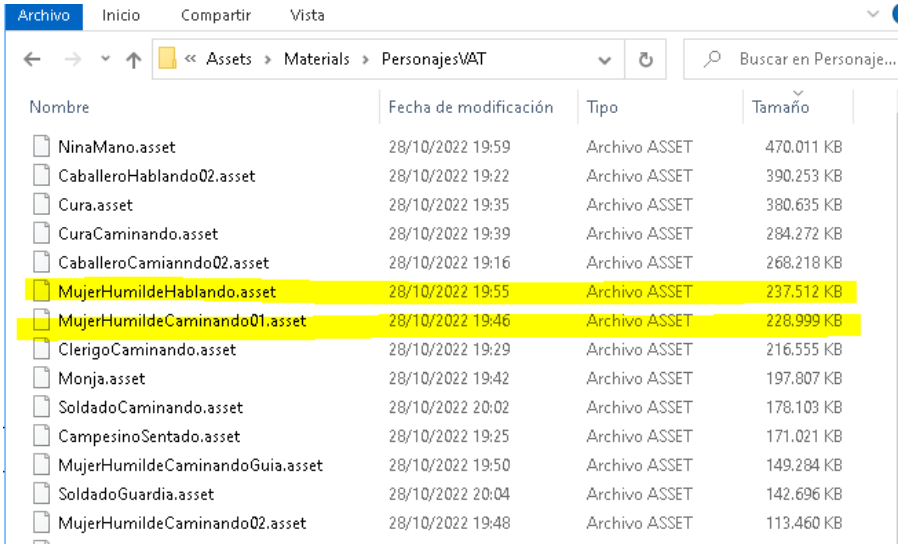


***Ilustración 2**. Tamaño assets VAT*

- **URP.** El pipeline utilizado en el proyecto es el integrado, mientras que, en las nuevas versiones de Unity, hay dos nuevos pipelines gráficos, el URP (Universal Render Pipeline) y el HURP (High Universal Render Pipeline), ambos con unas mejoras en cuanto a rendimiento gráfico muy superiores, así como características de postprocesado ya integradas en la propia API. Debido a que HURP no funciona en plataformas móviles, incluyendo los dispositivos de realidad virtual, se intentará migrar el proyecto a URP.

- **Dependencia de la Meta Oculus.** Se comprueba que el estado actual está desarrollado utilizando los *plugins* y paquetes

específicos de Oculus Quest, haciendo más complicado su compatibilidad con otros dispositivos VR.

1.4 Objetivos finales

Tras lo explicado anteriormente, finalmente los objetivos concretos de este trabajo serán los siguientes:

1. Disminuir el tamaño en conjunto del proyecto, otorgándole más estabilidad y disponibilidad, concretamente, se intentará evitar que ningún fichero ocupe más de 50 MB, límite del proveedor GitHub en su versión gratuita, que es el sistema de control de versiones distribuido basado en git, elegido para este trabajo. Esto facilitará el funcionamiento colaborativo en un futuro a través de este popular sistema de control de versiones.

2. Mejorar el occlusion culling aplicado al terreno.

3. Crear varias mallas por personaje con diferente número de vértices para poder aplicar un nivel de detalle, "*Level of Details" (LOD)*

4. Utilizar la nueva APIs gráfica, utilizada con el nuevo pipeline gráfico URP de Unity.

5. Utilizar la arquitectura intermedia desarrollada por Unity en sus últimas versiones, denominada XR, para el manejo de sistemas de realidad virtual, aumentada y mixta. Esto permitirá utilizar la aplicación con prácticamente cualquier dispositivo XR del mercado sin modificar el código.

6. Eliminar la dependencia con el software Vertex Animation Tools (VAT). El principal motivo para intentar esta iniciativa se basa en su incompatibilidad con versiones de Unity superiores a la 2019.4.9.f1, restringiendo la posibilidad de futuras actualizaciones y escalado de este proyecto.

Además, su condición de software *de pago,* y el ya comentado uso de *assets* con un tamaño demasiado grande, convierten a este objetivo, en algo esencial.

7. Conversión del proyecto hacia el motor de videojuegos Unreal Engine, actualmente en su versión 5.x

Capítulo 2
Desarrollo del Proyecto

2.1 Comenzar el proyecto Unity desde cero.

Como primera medida se decide comenzar un ***proyecto nuevo desde cero***, aprovechando lógicamente el material realizado por los desarrolladores gráficos, como son los personajes, objetos, terrenos y casas. Para esto se dispone de los modelos originales con los que comenzar a trabajar. El material viene en formato de la aplicación de software gratuito y código abierto Blender, junto a las animaciones de la tela en formato *"point caché"* tipo MDD, proveniente del software de creación de vestimentas Marvelous Designer.

Esta decisión fue tomada en base a las siguientes consideraciones:

- La eliminación de la dependencia con la herramienta *Vertex Animation Tools (VAT)*, ya nos permite una actualización de la versión del motor.
- Las características del nuevo pipeline gráfico URP, también requieren de una actualización de la versión del motor, que ahora es posible al no depender de VAT. Aunque se puede migrar hacia URP un proyecto creado con el pipeline gráfico estándar de Unity, la mejor opción es iniciarlo desde cero, ya que se pueden producir problemas en la transformación de materiales y texturas.
- Para lograr hacer compatible la aplicación con una amplia gama de dispositivos de realidad virtual, también se necesita actualizar la versión del motor, para disponer de la arquitectura intermedia que nos lo permite. [2.5]

2.2 Primer paso. Universal Render Pipeline (URP)

Se comienza el proyecto desde cero, inicialmente con la versión del motor de Unity la 2022, aunque posteriormente es actualizado a la 2023 sin presentar ningún inconveniente. Se crea usando las plantillas ya definidas para el aprovechamiento de URP [4].

De esta forma, toda la aplicación, pasará a utilizar este nuevo conjunto de APIs optimizadas y salvo algunos problemas con la importación de materiales que usan texturas como mapa de normales, aparentemente no se encuentran más inconvenientes que los propios del aprendizaje de su configuración óptima inicial.

Con este nuevo pipeline gráfico, las mejoras obtenidas de forma inmediata incluyen el rendimiento y la calidad gráfica. Se puede consultar una comparativa de prestaciones en la referencia [5].

Cabe destacar, que aún hay un framework superior, el HDRP, pero que sin embargo no está disponible para dispositivos móviles, como es el caso de los dispositivos HMD que aborda este trabajo, por lo que su utilización tuvo que ser descartada.

2.2 Animación de personajes. Vertex Animation Tools (VAT)

Este paquete era el principal escollo para poder realizar la actualización hacia las nuevas versiones del motor, ya que es el encargado de las animaciones de los personajes. Al disponer de animaciones no solo del esqueleto, sino también de la ropa, realmente fusionar ambos movimientos en una serie de *frames*, desde el punto de vista del rendimiento en un motor de videojuegos, es una buena práctica, ya que, de otra forma, los cálculos necesario

para el movimiento del esqueleto junto a la ropa, consumiría demasiados recursos como para poder realizarlos en tiempo real.

Esta animación por vértices se basa en que, a partir de una malla principal, normalmente denominada *pose*, solo almacenando los cambios de posición de cada uno de sus vértices, ya sea de forma relativa o absoluta, se consigue simular un movimiento, al estilo de la tradicional animación 2D, basada en una secuencia de *fotogramas/frames*. A la información que indica la nueva posición de cada fotograma, por cada uno de los vértices, se le llama, en la nomenclatura de Unity, Blend Shapes y realmente es una cualidad que ya incorpora el *componente Skinned Mesh Renderer* de Unity y de la que la utilidad VAT hace uso.

Las principales tareas resultantes de aplicar VAT, son, por un lado, las de crear un *asset* que incorpore esos Blend Shapes necesarios para crear una animación, y por otro, la de la animación en sí.

Sin embargo, el *asset* resultante creado por esta herramienta, resulta en un fichero demasiado pesado, y la animación requiere de un script para su funcionamiento, ya que no cuenta con la posibilidad de utilizar el *componente Animator* de Unity, lo que limita las características de animación que incorpora el propio motor de videojuegos.

2.2.1 Animación sin VAT. Animación con Blender

Gracias a Blender, herramienta de edición gráfica de software libre y código abierto, partiendo de los ficheros originales, se consigue recrear todos los Blend Shapes necesarios (llamados Shape Keys en la nomenclatura Blender), así como integrar varias animaciones distintas, y todo ello en un único fichero FBX por personaje.

Se obtiene un fichero del personaje con la misma calidad y con todas las animaciones juntas, sin depender de la aplicación VAT.

Además, el resultado es de tan solo una mínima porción del tamaño original, no superando los 50 MB que se impuso como objetivo, no solo por rendimiento, sino también para su utilización con Git junto con el proveedor GitHub de Microsoft.

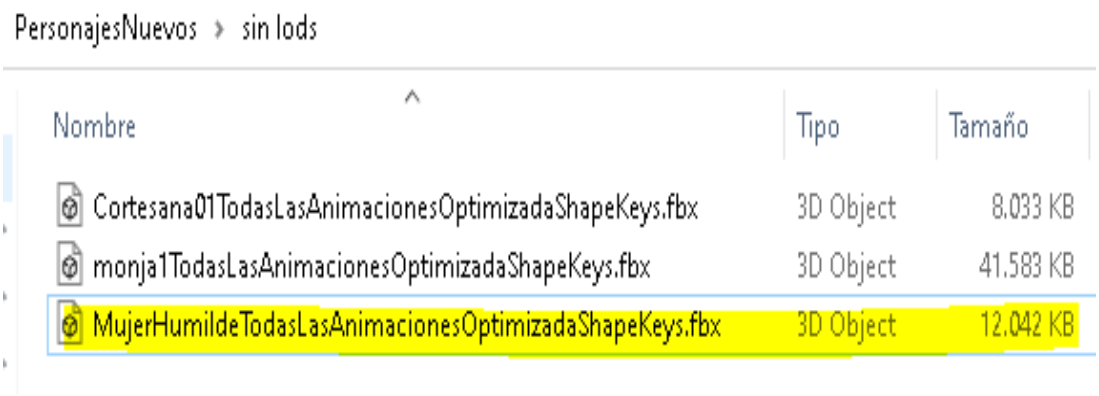


***Ilustración 3**. Nuevos ficheros de personajes*

Se puede observar claramente, como se han transformado los casi 500 Mb de la suma de las dos animaciones del personaje *"MujerHumilde"*, *[ilustración 2]* a un único fichero con sus animaciones incorporadas de tan solo 12Mb.

Cabe destacar, que, a partir de este momento, las animaciones son gestionadas ya por el *complemento Animator de Unity,* no teniendo ahora ninguna limitación en cuanto al tratamiento de las mismas.

El proceso exacto para realizar esta conversión se detalla en el Anexo I.

2.3 Renderizado. (Estudio del Occlusion Culling)

El *Occlusion Culling es una t*écnica muy utilizada y realmente se adapta muy bien para este tipo de aplicaciones, en las que las propias

edificaciones ayudan a ocultar, y no tener que renderizar, todo aquello que queda ocluido por otros objetos. Esto se traduce en una gran mejora en cuanto a rendimiento, al no tener que renderizar todo el *frustum de la cámara* (todos los objetos interceptados por el campo de visión)

Sin embargo, al funcionar esta técnica solo con objetos completos, hay que buscar un equilibrio entre la cantidad de fragmentos en las que podemos dividir una malla, con el objetivo de aplicar *occlusion culling* y el rendimiento obtenido. De hecho, en este caso, nuestro terreno está fragmentado en más de 6.000 mallas, como ya se indicó en el *apartado* [1.3](), creando sus consiguientes 6.000 GameObjects de Unity.

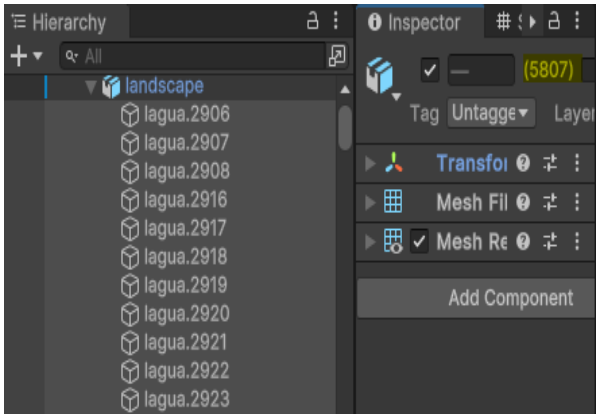


***Ilustración 4**. Resaltado en amarillo, 5807 GameObjects conformando el terreno.*

Comprobando simplemente los *frames* por segundo de una escena formada tan solo por este terreno fragmentado y una cámara desde una distancia fija, primero de lejos y después de cerca, se observa que, en el mejor de los casos, la tasa máxima de frames por segundo alcanzada es de 306 fps. Como es de esperar, este rendimiento se consigue cuando la cámara está cerca, en el que el algoritmo del *recorte (clipping)* prácticamente dejará de renderizar la mayoría de las mallas.

Al realizar una prueba sencilla consistente en unir el terreno en una única geometría y aplicarle el mismo escenario de pruebas, se obtienen resultados que demuestran un aumento de *frames* por segundo considerable. Así que en este caso concreto, la aplicación

del *occlusion culling* es contraproducente, por lo que se deshabilita esta técnica solo en el caso del terreno.

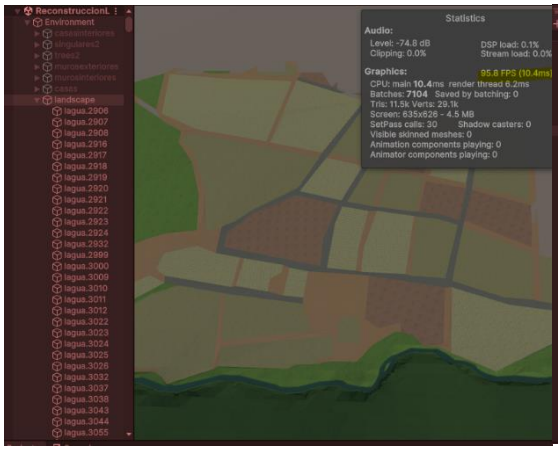
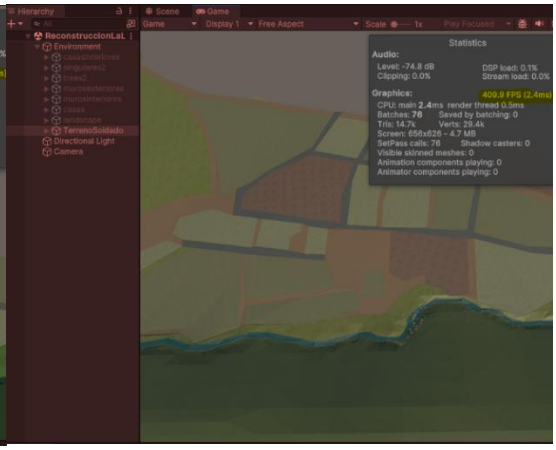
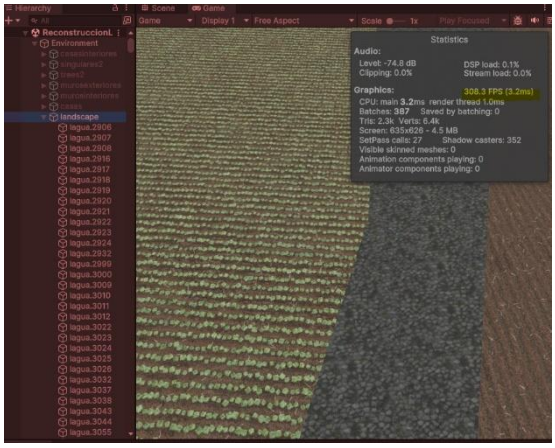
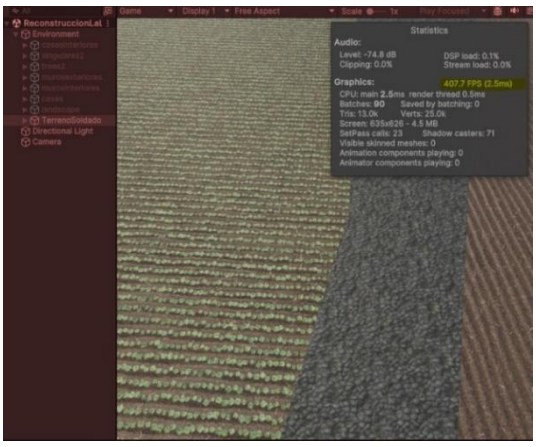
Terreno Fragmentado (cámara lejana)	Terreno Soldado (cámara lejana)
	
Terreno Fragmentado(cámara cercana)	Terreno soldado(cámara cercana)
	

***Tabla 1**. Terrenos fragmentados (Izquierda), Terrenos soldados (derecha)*

Se resume las cuatro imágenes anteriores en la **tabla 2**.

Rendimiento frames por segundo(fps)	**Terreno Fragmentado**	**Terreno Soldado**
Distancia de cámara (CERCA)	**306 fps**	**409fps**
Distancia de cámara (LEJOS)	**95 fps**	**407 fps**

***Tabla 2**. Resumen comparativo del terreno dividido y soldado en frames por segundo*

Se puede observar como el rendimiento no varía cuando la cámara está cerca o lejos en el caso del terreno soldado, lo cual es el comportamiento esperado, ya que ahora el terreno es una única malla. La mejora obtenida es de un 33% en el peor de los casos, pasando de aproximadamente 300 a 400 *frames* por segundo con la visión desde cerca.

Se consigue mejorar aún más el comportamiento del terreno soldado como una única malla, dividiendo su geometría en 8 partes, lo que permite dejar de renderizar partes del terreno que queden fuera de la visión del jugador, pero no por el *occlusion culling*, sino porque esas partes del terreno están afectadas por el *clipping,* ya que si están más allá del plano lejos (distancia máxima de renderizado) de la cámara o fuera del ángulo de visión establecido, se ven afectados por el algoritmo de recorte contra el volumen de vista.

Con esta combinación, se consigue un buen equilibrio entre número de mallas y las ventajas del *occlussion culling*.

Por si fuera poco, la suma de los ficheros de ocho terrenos, ocupan solo 1.38 Mb en total, frente a los 23.3 Mb que ocupaba el fichero del terreno original, fragmentado en más de 6.000 partes. En ambos casos, estos tamaños no incluyen las texturas asociadas.

2.4 LOD. La optimización estrella.

Sin lugar a dudas, la técnica que mejor resultados ofrecería en cuanto a optimización se refiere, sería aplicar Level of Details (LOD por sus siglas en inglés), o sea, disponer por cada personaje de diferentes calidades de mallas en cuanto a número de vértices, lo que permitiría recrear escenas más realistas sin sobrecargar el hardware, ya que, en principio, habría muchos personajes, caminando o hablando lejos de la cámara, siendo poco visibles y candidatos perfectos a ser renderizados con una malla más simple. En los personajes originales, la calidad de las geometrías es de 100.000 triángulos de media, llegando incluso a más de 600.000 en el caso de algunos de ellos. Esta cantidad de triángulos y vértices es inasumible para un hardware *standalone*, es por eso por lo que en una optimización anterior se implementó un patrón de diseño conocido como *"Object Pooling"* [8], que permitió limitar el número de personajes activos en la escena de manera simultánea.

Lamentablemente, los personajes disponibles no traen esta variedad de mallas adjuntas, o sea, no están preparadas para aplicar LOD. Hay que tener en cuenta que el LOD es normalmente desarrollado por el equipo de desarrollo gráfico, así que la solución, desde el punto de vista informático, consiste en reducir la cantidad de vértices de los personajes, utilizando una técnica llamada *retopología*. Sin embargo, hay dos problemas al aplicar esta técnica en el caso de estos personajes animados.

Por un lado, la animación del cuerpo se realiza mediante esqueleto, esto significa que hay huesos conectados a vértices en concreto de la geometría, los cuales podrían desaparecer en el retopologizado, teniendo que ser reasignados nuevamente.

Por otro lado, la animación de la tela es independiente, basada en animación por vértices, *point caché*, cuyo funcionamiento es

completamente diferente al procedimiento por esqueleto. En este caso, a partir de una malla inicial, a la que llamamos *pose*, se almacenan la variación o nueva posición de cada uno de sus vértices en un instante de tiempo, lo que obliga a que los *point cache* tengan la misma cantidad de vértices que la malla inicial o pose. Por eso, si retopologizamos la pose de la tela, también habría que recrear nuevamente todo el movimiento de esta, ya que al haber disminuido la cantidad de vértices, ahora no concuerdan con los de los point cache, debiendo ser creados de nuevo.

Claramente, estas dos operaciones darían el mejor de los resultados, pero tendría que ser una tarea propia de un equipo de diseño.

2.4.1 Optimización con Blender

Con Blender, se consiguen dos optimizaciones importantes.

- **Optimización del Cuerpo:** Dado que la mayoría de los personajes llevan indumentarias que ocultan la mayor parte del cuerpo, se eliminan todos los vértices que no son visibles debido a la ropa, incluyendo en algunos casos incluso los pies. En la siguiente tabla se puede observar el resultado de esta *optimización en crudo* de los personajes.

Personajes	Vértices / Triángulos Iniciales	Vértices / Triángulos Resultantes
Caballero caminando	162.018V - 284.443Tris	143.517V - 249.310 Tris
Cortesana 01	44.251V - 86.976 Tris	40.968V - 80.256T
Cortesana 02	58.050V - 104.937 Tris	48.292V - 85.699 Tris
Monja	332.875V – 657.430 Tris	239.324V – 470.110 Tris
Mujer Humilde	74.676V – 132.710 Tris	52.751V – 91.757 Tris

***Tabla 3**. Número de vértices(V) y triángulos (Tris) antes y después de la reducción del cuerpo.*

En la siguiente imagen se muestra el aspecto de los personajes, antes y después de la eliminación de vértices no visibles.

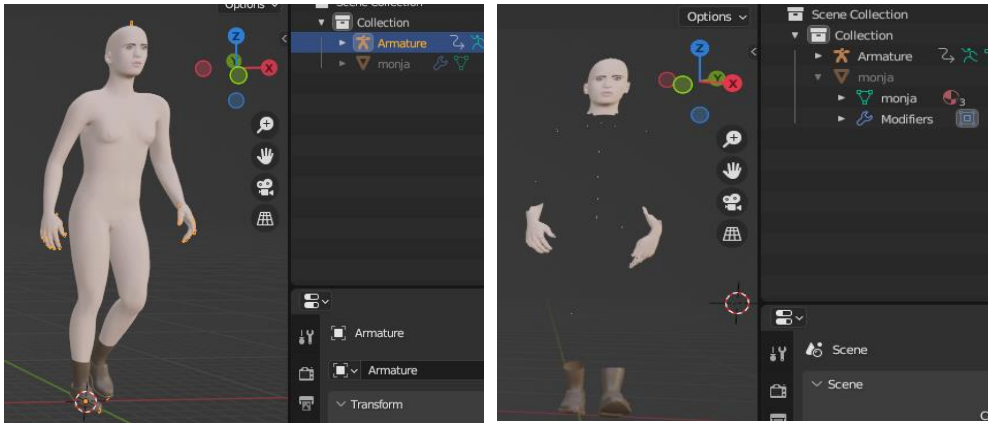


***Ilustración 5**. Antes y después de eliminación vértices ocultos*

- **LOD.** A partir de los personajes creados en el punto 2.2.1, y ya con la optimización del cuerpo realizada en el punto anterior, obtenemos una única malla que contiene el aspecto del personaje, incluida su ropa, en cada uno de los frames de su animación. Ahora, gracias al proceso de retopologización, se

crean mallas derivadas con diferentes cantidades de vértices, y como se trata de una única geometría, no es necesario sincronizar el movimiento de la ropa y el cuerpo.
La cantidad de vértices de la malla original se respeta en todos los personajes, y después se crean tres niveles más de LOD por cada una, lo que permitirá posteriormente disponer de bastante versatilidad y diferentes combinaciones de distancia/calidad desde Unity. Para todos los casos, la cantidad de vértices resultantes en cada nivel de detalle (LOD), será un porcentaje de los vértices de la malla original con la optimización del cuerpo del punto anterior ya aplicada.

En la siguiente tabla, se muestra la cantidad de vértices definitiva de los que se compone cada nueva malla en cada nivel de detalle, pudiéndose comparar la reducción de vértices fácilmente con respecto a la malla original.

Personaje	Malla Original 100% de los vértices (aplicada la optimización del cuerpo)	40% De los vértices	10% De los vértices	0,005% De los vértices
Caballero Caminando	**143.517 V 249.310 Tris**	61.080 V 99.724 Tris	19.544 V 24.930 Tris	6.318 V 1.246 Tris
Cortesana 01	**40.968 V 80.256 Tris**	16.758 V 32.101 Tris	4.410 V 8.024 Tris	285 V 401 Tris
Cortesana 02	**48.292 V 85.699 Tris**	20.687 V 34.279 Tris	6.072 V 8.569 Tris	1.100 V 427 Tris
Monja	**239.324 V 470.110 Tris**	98.084 V 188.043 Tris	26.288 V 47.011 Tris	2.700 V 2.349 Tris
MujerHumilde	**52.751 V 91.757 Tris**	23.740 V 36.701 Tris	8.301 V 9.175 Tris	3.350 V 458 Tris

Tabla 4. *Calidad de los dierentes niveles LODs.*

Cabe destacar, que la incorporación de LODs a los diferentes personajes, al realizarse sobre el mismo fichero FBX, incrementa lógicamente el tamaño de este, pero se sigue manteniendo el compromiso de no superar los 50 MB por fichero, de hecho, los tamaños siguen estando bastante por debajo de ese límite. Por ejemplo, se puede apreciar en la ilustración 6, como la MujerHumilde, ocupa 19 MegaBytes con los niveles de LODs ya aplicados, mientras que en su versión sin niveles de detalle ocupa solo 8 MegaBytes menos, como se puede apreciar en la ilustracion 3.

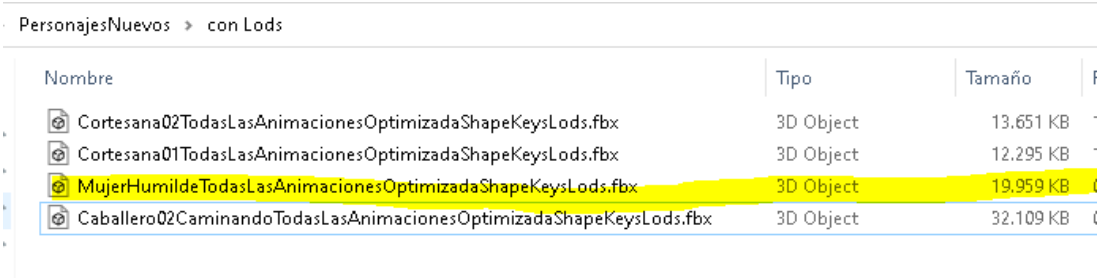


Ilustración 6*. Tamaño final de los FBX con optimización de cuerpo, animaciones y Lods aplicados.*

Todo el proceso para aplicar retopología, se explica detalladamente en el Anexo II.

2.5 Compatibilidad con otros HMD.

Para aislar la programación del sistema de Realidad Virtual (VR), aumentada (AR) o mixta (XR), tecnologías que Unity engloba bajo el término XR, el motor incorpora una capa de abstracción que hará de intermediario entre los diferentes dispositivos y la implementación realizada por el programador. Esta arquitectura intermedia, se basa en *plugins* denominados XR SDK, cuya API es multiplataforma y con la que se intenta reducir el esfuerzo para soportar nuevos dispositivos. Todo ello es accesible a través del paquete *XR Plugin Management,* y está íntimamente ligado a la

versión de Unity utilizada, es por ello, que, ya que ahora es posible actualizar las versiones del motor en este proyecto, se puede utilizar la versión más actual de este paquete, en concreto la 4.3.3, disponible a partir de la versión 2022.3 del motor. [6]

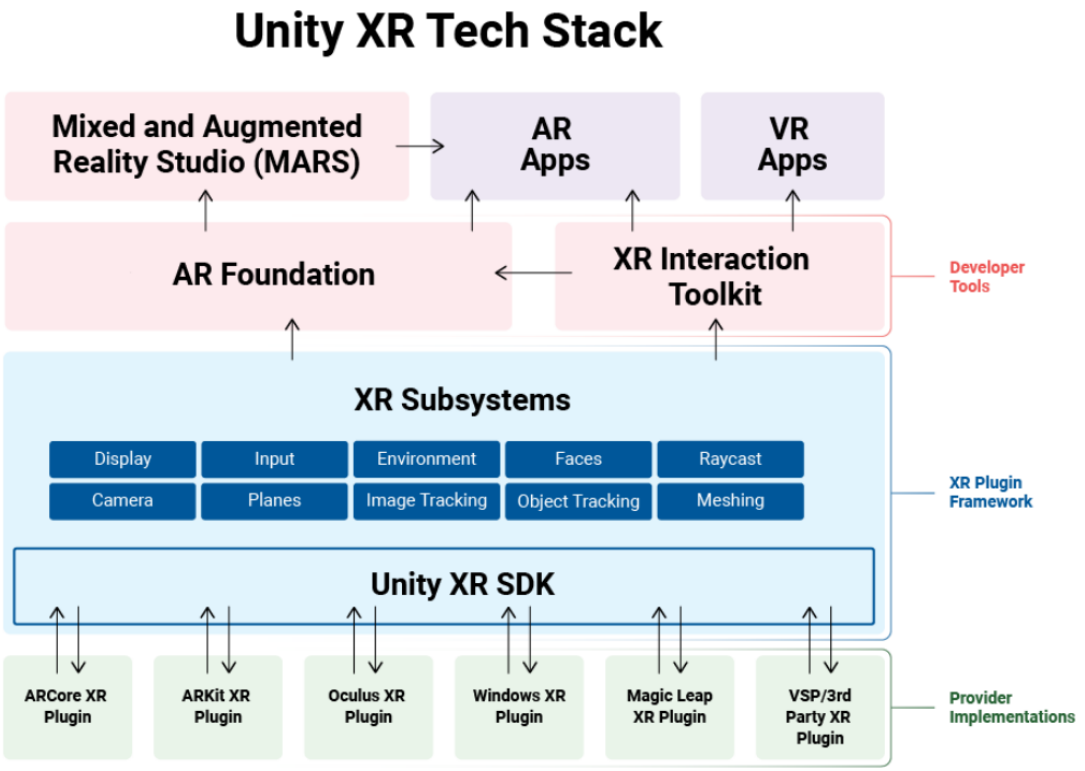


***Ilustración 7**. Arquitectura XR de Unity.*

Fuente: https://docs.unity3d.com/Manual/XRPluginArchitecture.html

Con este sistema, son los fabricantes los que implementan lo *drivers* de las capas inferiores para sus dispositivos, bien utilizando controladores propios o adaptándose al estándar Open XR [11][12]. El desarrollador se limitará a utilizar las funcionalidades de las capas superiores.

Esto no quiere decir, que no sea posible utilizar capacidades propias de un HMD en concreto accediendo a su propia API de más bajo

nivel, pero eso rompería la compatibilidad con el resto de los dispositivos.

2.6 Estructura del proyecto.

El resultado final del proyecto es el de la recreación de la Plaza del Adelantado con sus edificios singulares y alrededores. En la imagen inferior, se muestra una captura de toda la geometría del proyecto.

***Ilustración 8**. Proyecto completo. Plaza del Adelantado y alrededores.*

Específicamente la Plaza del Adelantado, con un escenario inmersivo, representando un día de mercado.

***Ilustración 9**. Plaza del Adelantado. Día de Mercado.*

La estructura del proyecto queda representada en la siguiente imagen. Se muestran tanto las carpetas del proyecto, como su estructura en el inspector de Unity.

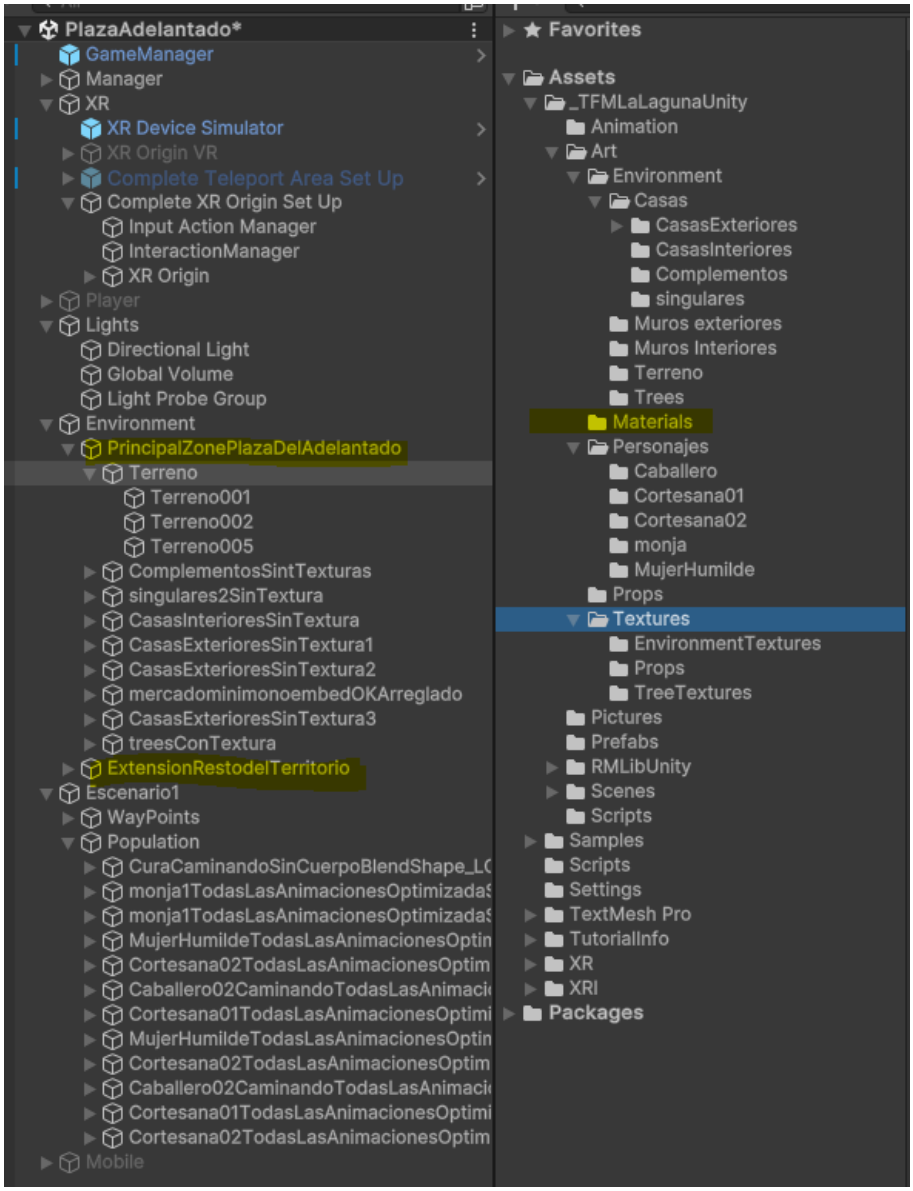


***Ilustración 10**. Estructura del Proyecto. Inspector Unity(izquierda), organización carpetas en disco(derecha)*

Por otra parte, se dispone de una carpeta de recursos, en la que se encuentran todos los objetos disponibles después de haber aplicado las distintas optimizaciones, como texturas, LODs o shapekeys.

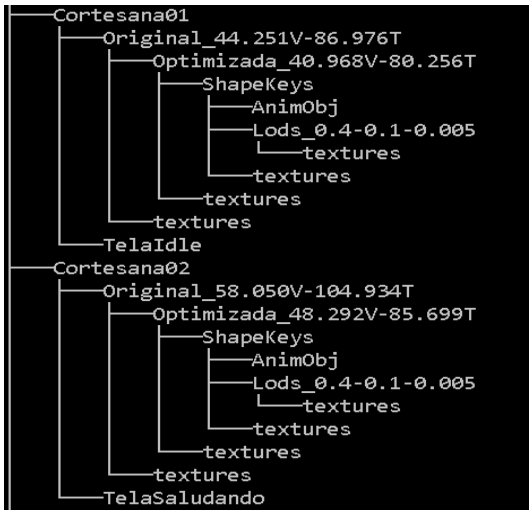


***Ilustración 11**. Aspecto estructura carpeta recursos, sección personajes.*

Capítulo 3 Mejoras Específicas

3.1 RMLib, librería NPC.

Con respecto a la recreación de escenas realistas, se incorpora una librería de creación propia especializada para el control de personajes no jugadores, NPC, a la que he llamado RMLibUnity.

Esta librería es un paquete independiente que se puede agregar a cualquier proyecto Unity, dotando no solo de capacidades de inteligencia artificial a los NPCs, sino que además facilita el control de otras características del gameplay.

Está basada en la técnica de acciones basadas en objetivos o GOAP[9], que funciona asignando a cada personaje diferentes planes. Cada plan está formado por varias acciones hasta llegar a una acción objetivo, en cuyo caso se generará un nuevo plan. Así mismo, cada personaje puede tener varios planes asociados y estos pueden tener acciones dependientes de cualquier estado del juego o acciones completamente independientes. Los objetivos no tienen por qué ser únicamente movimientos de un lugar a otro, podrían ser tareas de cualquier tipo, además estas tareas podrían ser interrumpidas por otras de mayor prioridad, provocar que se buscara un plan alternativo o bien que se continuara con la acción siguiente del plan en curso. La flexibilidad es enorme, tanto que incluso los movimientos de los personajes también son gestionados por GOAP, y no por una máquina de estados del animador de Unity.

Entre otras ventajas, en cuanto a movimiento de los personajes, se encuentra la posibilidad de buscar caminos alternativos, detectar colisiones y recalcular rutas, permitir movimientos con rotaciones más realistas evitando rotar sobre sí mismo mientras sigue una línea recta hacia el siguiente destino y mucho más.

La lógica de los NPC les permite tener acceso al estado del mundo en cualquier momento, y por ello es posible que la interacción de los personajes con el jugador se pueda hacer sin necesidad de utilizar costosos *raycast* junto a sus costosos *colliders*. En su lugar, simplemente se sabe exactamente la posición y ángulo del jugador con respecto a cualquier personaje, y así este puede responder en consecuencia. En la **ilustración 11**, se muestra un diagrama de clases resumido.

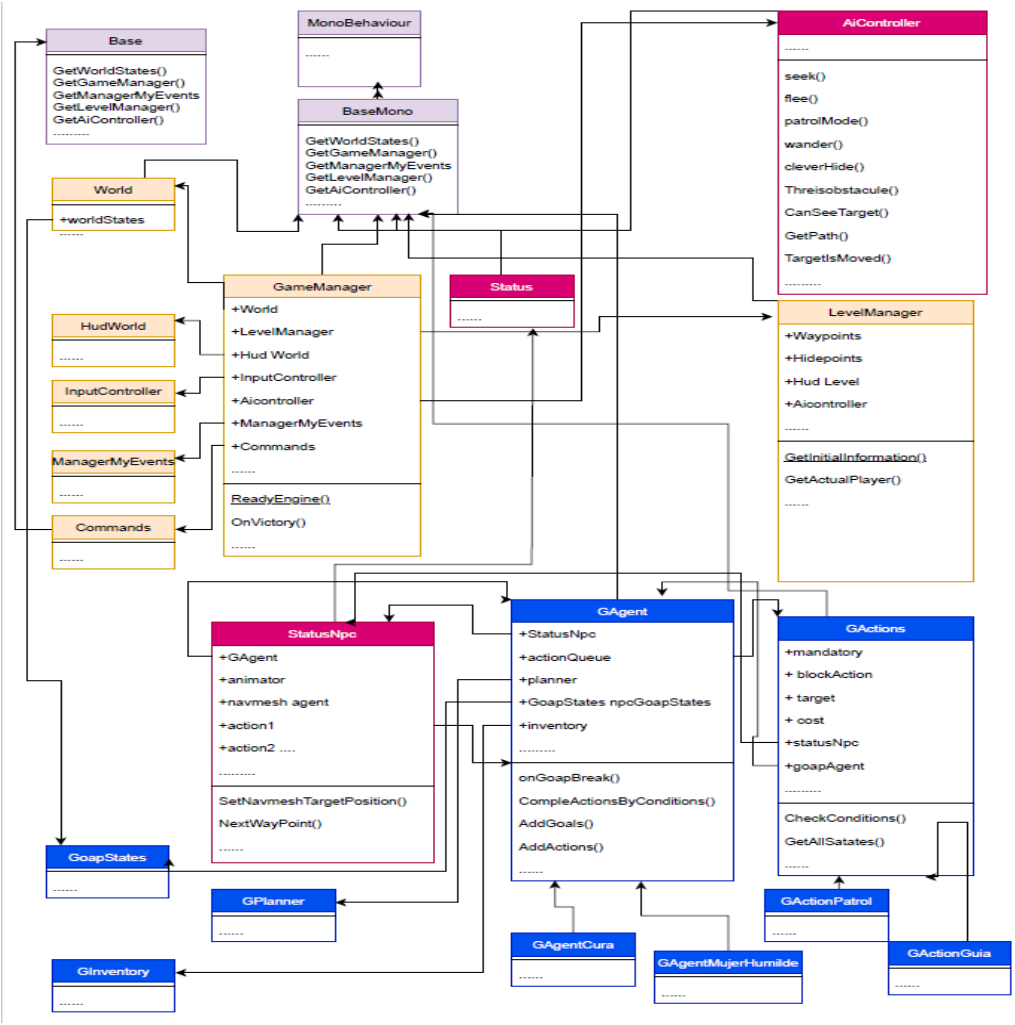


***Ilustración 12**. Diagrama de clases resumido de RMLibUnity*

Las clases en color marrón claro, dotan de características de control, como un ***GameManager***(único durante todo el juego), un ***LevelManager*** (propio por cada escena), eventos y mucho más.

Las clases en color rosado, representan la inteligencia artificial, cada NPC contendrá un complemento ***StatusNpc*** o derivado de este si se quiere ampliar o personalizar.

La clase ***AIController*** contendrá todos los métodos relacionados con la inteligencia artificial, y solo existiráuna instancia en memoria, creada por la clase ***GameManager***. ***AIController*** recibe como parámetro el ***StatusNpc***, del que obtener toda la información requerida para el funcionamiento de sus métodos.

Las clases en azul, representan el conjunto de clases asociadas a la planificación de acciones orientada a objetivos (Goap). Básicamente, cada NPC tendrá un único complemento heredado de la clase ***GAgent*** y varias instancias heredadas de la clase ***GAction***, que definirán las acciones. Estos complementos junto al ***StatusNpc*** indicado anteriormente, forman el conjunto de clases necesarias para la IA. También se dispone de la clase generadora de los planes para cada personaje(***GPlanner***) así como la posibilidad de disponer de un inventario de objetos con ***GInventory***.

En las **ilustraciones 12 y 13**, se muestra el aspecto de algunas configuraciones posibles, independientes para cada NPC.

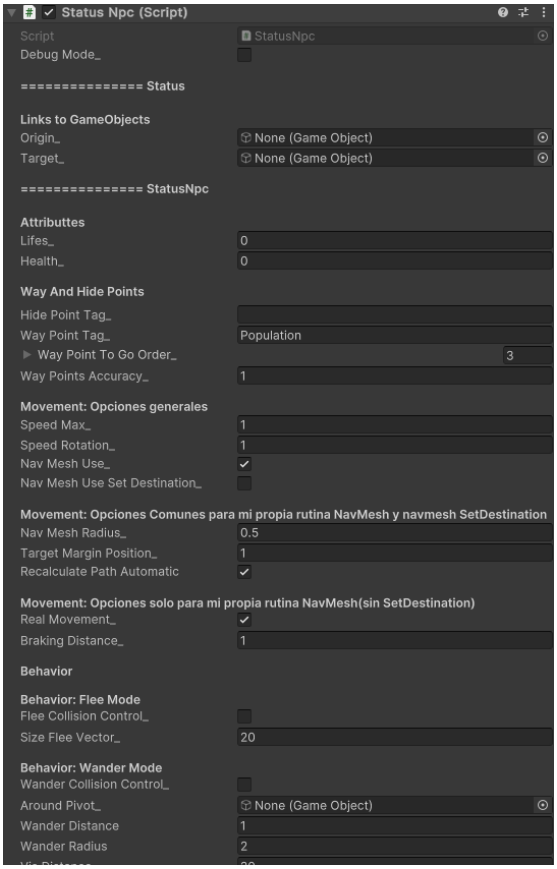


***Ilustración 14**. Aspecto y algunos valores para personalización de StatusNpc.*

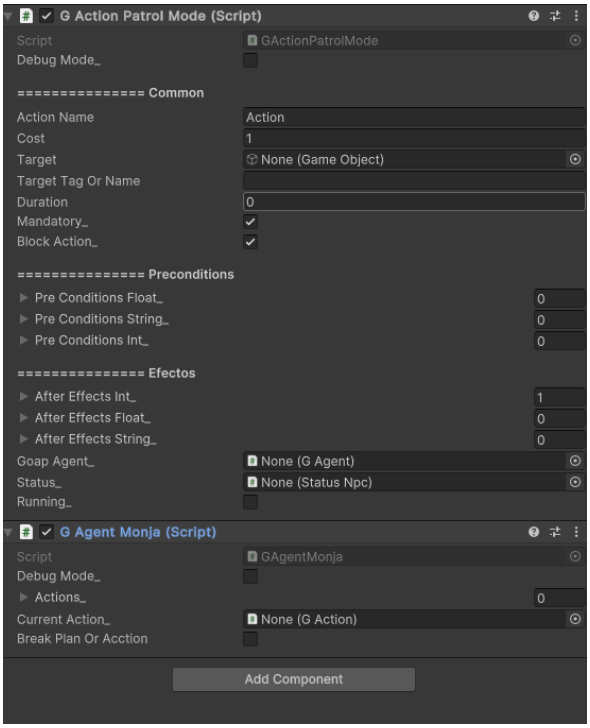


***Ilustración 13**. Componente Agente Goap agregado a NPC junto con un componente acción.*

3. 2 Optimizaciones varias.

En cuanto a importación de modelos, se han mejorado y reparado las siguientes cuestiones:

- **Árboles**
  La importación de los árboles no se muestra correctamente, ya que sus hojas aparecen cuadradas y opacas. Esto es debido a que los materiales que aplican texturas con recorte, llamadas *clipping,* no son importadas con esta

característica activada por parte del motor de Unity. La solución es fácil, simplemente en el material asociado a la malla, debe de activarse la opción de "*Alpha Clipping*" y adaptar el valor de "*Threshold*" para obtener el aspecto deseado. Adicionalmente, se puede modificar la propiedad "*Alpha Is Transparency*" que se encuentra en las opciones de la textura asociada para buscar el efecto deseado. En las ilustraciones 15,16 se muestran las opciones afectadas y en las ilustraciones 17 y 18 el resultado del antes y después del aspecto de los árboles.

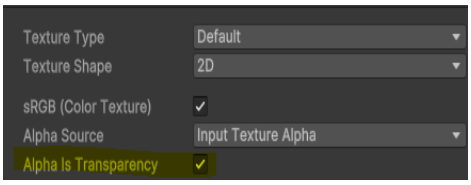


***Ilustración 15**. Modificación de Alpha en texturas.*

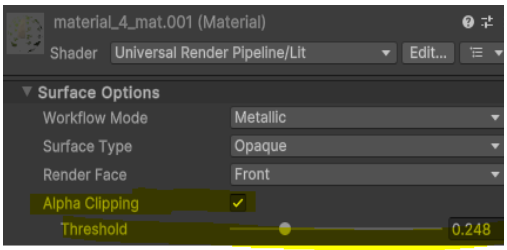


***Ilustración 16**. Activar Clipping en materia*

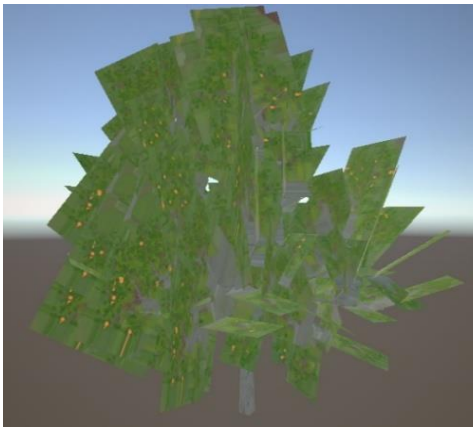


***Ilustración 17**. Aspecto proyecto original de los árboles*

***Ilustración 18**. Aspecto árboles corregidos.*

- **Texturas**

  Unity, genera un mipmaps por cada textura, con el fin de optimizar estas según su distancia. Podríamos decir que es una técnica similar a LODs, pero aplicado a las texturas y no a mallas. Aun así, se ha reducido el tamaño de los ficheros originales de las mismas reduciendo su resolución. Esto resultó, en un menor tamaño del proyecto en general y por supuesto, un menor tamaño por cada fichero de texturas, a costa de una inapreciable pérdida de calidad. Esto fue una sorpresa, pero parece debido a que la resolución inicial de las texturas, 4096 * 4096 en la mayoría de los casos, era ya excesiva, por eso, su conversión a una resolución de 256*256, no provocó una pérdida de calidad aparente.

- **Personajes levitando**

  Para el movimiento de los personajes por el escenario, se crea una malla de navegación, llamada en el caso de Unity, *NavMesh Navigation*. Esta malla indica los lugares por los que es posible que pueda caminar cualquier personaje. Con este terreno, este *NavMesh* Navigation creado por Unity, se eleva unos centímetros por encima del suelo, por lo que no se ajusta completamente con la malla del terreno, como cabría esperar, presumiblemente debido a los desniveles de este. Esto provoca que los personajes parezca que levitan sobre el suelo.

  Averiguado el motivo, se soluciona fácilmente ajustando el valor de la variable *"Base offset"* del *componente "Nav Mesh Agent"* de cada personaje. Con un valor de **-0,15 metros,** se compensa el desajuste entre las mallas del terreno y la malla del *navmesh* generado.

- **Sin Colliders**
  Con la implementación actual, gracias a la librería de NPCs y aprovechando los propios límites de las mallas de navegación*(navmesh)*¸ se ha podido implementar todo el *gameplay* sin necesidad de agregar ninguna malla de colisión(colliders) a ningún objeto, con el consiguiente ahorro de recursos.

- **Tamaño**

  Con el propósito de reducir aún más el tamaño general del proyecto y de evitar contener ningún fichero superior a 50 Mb, se realizaron las siguientes optimizaciones.

  - Los antiguos ficheros **. assets* fueron convertidos a **.fbx*.

  - A los nuevos ficheros *.fbx, se le extraen los materiales y las texturas.

  - Se crea una única carpeta dentro del proyecto que contiene todos los materiales, y se hace lo propio con las texturas. Esto no solo ayuda a la organización, sino que reduce el tamaño del proyecto en gran medida, ya que se detectaron muchos ficheros de materiales y texturas repetidos.

  - Los ficheros que ocupaban más de 50 MB, fueron divididos en varios, como por ejemplo el fichero *casas.fbx* original, que ocupaba 97 MB. Se muestra el resultado en la ilustración 19.

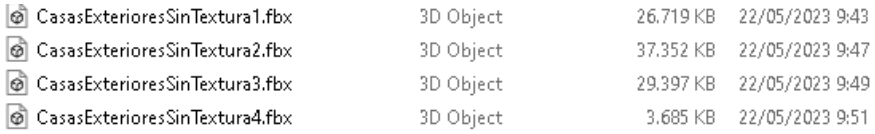


***Ilustración 19**. Casas.fbx dividido en cuatro ficheros nuevos.*

- **Otras configuraciones**

  Al ser un proyecto iniciado desde cero, se han importado y aplicado diferentes configuraciones ya utilizadas en los trabajos previos, como son el "Draw Call Batching", Compresión de texturas ASTC, baking de la iluminación y Occlusion culling para todos los objetos salvo el terreno.

Capítulo 4
Resultados

4.1 Objetivos

Con respecto de los objetivos inicialmente propuestos, el resultado final es el siguiente.

Primer Objetivo. (Disminuir el tamaño facilitando el desarrollo y haciéndolo compatible con git.)

El proyecto inicial sin compilar ocupa en disco alrededor de 15Gb, conteniendo además ficheros que superaban los 400 Mb, lo cual no permitía la colaboración a través de git. Finalmente, el tamaño final ha quedado en torno a los 350 Mb y si contener ningún fichero de tamaño superior al límite propuesto de los 50 Mb.

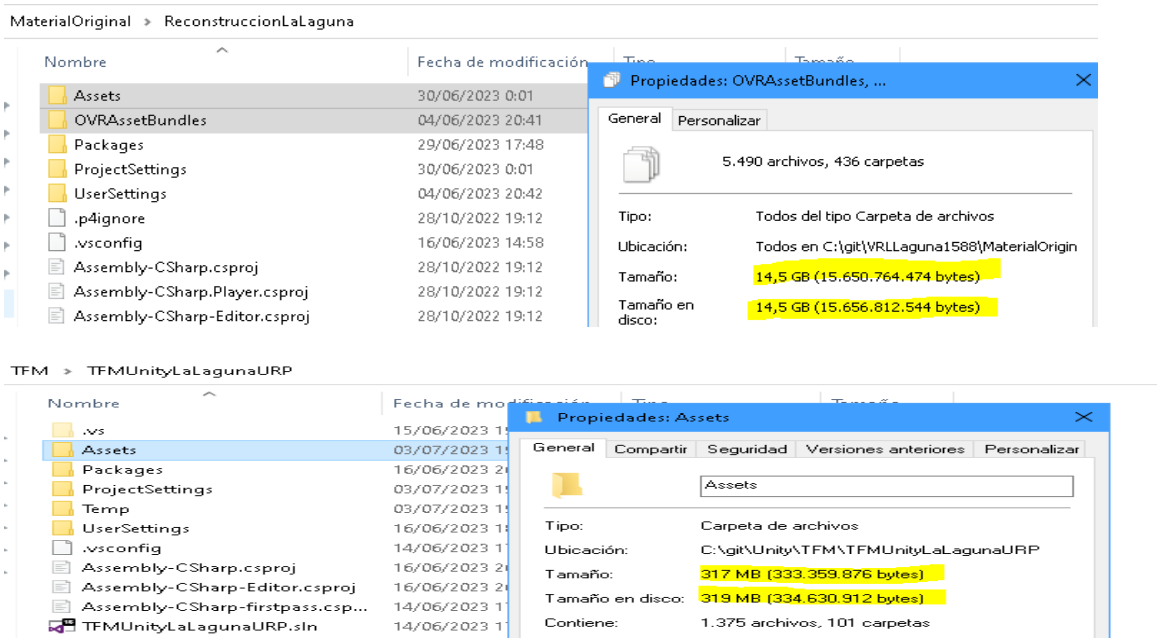


***Ilustración 20**. Tamaño inicial (Arriba), nuevo tamaño de proyecto (Abajo)*

Segundo Objetivo. (Mejorar el occlusion culling aplicado al terreno)

Dado el aumento en cuanto a rendimiento, y no solo en lo referente a frames por segundo, sino también a la disminución de recursos en general, creo que se ha logrado este objetivo gracias a la unificación del terreno.

La excesiva descomposición a nivel de *gameobjects* era uno de los causantes de bloqueos y esperas constantes en el editor de Unity, lo cual provocaba una iteración de desarrollo muy frustrante. Sin embargo, también se ha conseguido una buena mejora en este sentido.

Tercer Objetivo. (Level Of Details)

Sin duda, la mejor de las optimizaciones, ya que ésta no solo ha reducido drásticamente el tamaño en disco de los personajes, sino también se ha incrementado enormemente el rendimiento, permitiendo ahora poder crear escenas con más personajes activos de forma simultánea, y en definitiva causar una sensación más inmersiva. Hay un tutorial detallado de como poder adaptar cualquier personaje que se necesite en el *Anexo I*.

Cuarto Objetivo. (Universal Render Pipeline)

Disponer del proyecto ya migrado a este nuevo pipeline gráfico, no solo mejora el rendimiento, sino que garantiza la escalabilidad y compatibilidad futura del mismo, ya que se prevé que el actual pipeline gráfico deje de estar disponible en próximas versiones del motor de Unity.

Quinto Objetivo. (Compatibilidad con diferentes sistemas HMD)

En este punto, son las propias APIs del nuevo XR Plugin Manager, las que actuarán como intermediarias entre la aplicación y los dispositivos finales, siempre y cuando no se utilice ninguna funcionalidad de un HMD concreta.

Esta aplicación, se ha probado únicamente con el simulador integrado de Unity y el HMD de Meta Oculus Quest 2, ya que no se disponía de ningún otro hardware alternativo, sin embargo, están sentadas las bases, al no haber instalado drivers propios de ningún modelo en concreto, para que sea funcional con otros dispositivos HMD.

Sexto Objetivo. (Eliminar la dependencia del paquete Vertex Animation Tools - VAT)

Esta mejora ha resultado fundamental, ya que gracias a que se pudo cumplir este objetivo, se han podido cumplir también los objetivos número 4 y 5. También, se garantiza mayor escalabilidad y futuras ampliaciones y actualizaciones del proyecto.

Además, está el hecho de que ya no se depende de ningún software de pago y de la funcionalidad añadida que nos proporciona el poder disponer de animaciones utilizables desde el *componente animator* de Unity, pudiendo aprovechar tanto su máquina de estados como otras características.

Séptimo Objetivo. (Conversión a Unreal Engine 5)

Esta opción finalmente no se pudo explorar debido a la complejidad y tiempo empleado en los objetivos anteriores. Sin embargo, la eliminación del paquete VAT como requisito en la versión de Unity, sería un primer punto de partida, ya que en principio facilitaría bastante la conversión del proyecto, puesto que este paquete no está disponible para el motor Unreal Engine .

4.2 Conclusiones y Líneas futuras

4.2.1 Conclusiones

Aunque el proyecto original se encontraba en una etapa bastante avanzada y madura de desarrollo, la realidad es que la dependencia sobre el paquete VAT, así como los grandes problemas de rendimiento en dispositivos de realidad virtual en su modo de funcionamiento *standalone*, auguraban un futuro de este proyecto bastante incierto, ya que la posibilidad de actualizaciones y escalabilidad futura quedaban muy restringidas.

Tras este trabajo, se dispone de un proyecto limpio, desde cero, sin dependencias, funcionando con la última versión de Unity y con una mejora en cuanto a rendimiento bastante notable gracias fundamentalmente al LOD y al tratamiento del terreno aplicados.

Además, ahora es posible trabajar de forma colaborativa con el sistema de control de versiones distribuido *git*, y con un proceso iterativo de desarrollo mucho más ligero debido a que el nuevo proyecto tiene un consumo de recursos muy inferior en todos los sentidos, y todo ello sin pérdida de calidad.

4.2.2 Líneas Futuras

Debido a la recreación del proyecto desde cero, por cuestiones de tiempo, estaría pendiente la implementación de parte del *gameplay,* que si estaba disponible en las versiones anteriores, como por ejemplo las visitas guiadas a monumentos singulares, audios y algunas interacciones.

Otra innovación interesante, sería la de utilizar las características de interacción con objetos integradas en el paquete *XR Interaction ToolKit* de Unity, dando la posibilidad de poder manejar objetos,

extendiendo la inmersión más allá del simple movimiento y teletransporte que se usa actualmente.

Además, queda todavía explorar la vía inicial de la conversión del proyecto, desde el motor de Unity hacia el motor de Unreal Engine 5.

Finalmente, creo que, como mejora estructural y transversal de este proyecto, sería de gran utilidad el poder contar con un equipo de desarrollo gráfico, ya que en lo que respecta a dispositivos de bajo rendimiento como los HMD, pienso que se consiguen mejores optimizaciones con modificaciones y adaptaciones de la parte gráfica, que lo que se puede conseguir a nivel simplemente de programación.

4.3 Summary and Conclusions

4.3.1 Conclusions

Although the original project was at an advanced and mature stage of development, the reality is that the dependency on the VAT package, as well as significant performance issues on standalone virtual reality devices, cast doubt on the future of this beautiful project. The possibilities for updates and future scalability were severely limited.

After this work, we now have a clean project from scratch, free of dependencies, running on the latest version of Unity, and with a significant improvement in performance, mainly thanks to LOD and terrain optimization techniques applied.

Furthermore, it is now possible to work collaboratively using the distributed version control system Git, and the iterative development process has become much lighter due to the new project's significantly lower resource consumption in terms of both RAM and secondary memory, all without sacrificing quality.

4.3.2 Future research ines

Due to the project's recreation from scratch and time constraints, the implementation of some of the gameplay features, like guided tours of significant monuments, audio features, and some interactions that were available in previous versions should be rebuilt.

Another interesting innovation would be to take advantage of the object interaction capabilities integrated into Unity's XR Interaction ToolKit, allowing for object manipulation and extending immersion beyond simple movement and teleportation currently used.

Furthermore, there is still the possibility to explore the initial conversion of the entire engine to Unreal Engine 5.

Finally, having a graphics development team would be highly beneficial, as a structural and cross-cutting improvement to this project, Especially for low-performance devices like HMDs, I believe that better optimizations can be achieved through modifications and adaptations in the graphics department rather than relying solely on programming.

Bibliografía

[1] Merayo de Caso, Jonathan (2022) *<<Rutas patrimoniales en La Laguna, ambientadas en el siglo XVI, para dispositivos Oculus Quest 2. Optimización y actualización.>> Trabajo* Fin de Master por la Universidad de La Laguna, Spain

[2] Rodríguez Navarro, Cristian (2021) *<<Museo de vestimentas del s. XVI en Realidad Virtual>> Trabajo* Fin de Grado por la Universidad de La Laguna, Spain

[3] Fariña Barrera, Alberto (2022) *<<Rutas patrimoniales en La Laguna, ambientadas en el siglo XVI, para dispositivos Meta Quest 2. Gestión y Optimización de un proyecto en Realidad Virtual.>>* Trabajo Fin de Master por la Universidad de La Laguna, Spain

[4] Web oficial Unity (2023) https://unity.com/es/srp/universal-render-pipeline

[5] Web oficial Unity (2023) https://docs.unity3d.com/2023.2/Documentation/Manual/render-pipelines-feature-comparison.html

[6] Web Oficial Unity (2023) https://docs.unity3d.com/Manual/com.unity.xr.management.html

[7] Web oficial manual de Blender (2023) https://docs.blender.org/manual/en/latest/

[8] Object Poolling Wikipedia (2023) https://es.wikipedia.org/wiki/Object_pool_(patr%C3%B3n_de_dise%C3%B1o)

[9] Rafael Arnay del Arco, Master oficial Universitario en Desarrollo de Videojuegos, Inteligencia Artificial 2022/2023

[10] Web oficial Vertex Animation Tools(VAT) 2023

https://polyflow.xyz/content/vertex-animation-tools/documentation/index.html

[11] Web Oficial Open XR 2023 https://www.khronos.org/openxr/

[12] Web Wikipedia 20233. https://en.wikipedia.org/wiki/OpenXR

Anexo I. Personajes ASSETS a FBX

Hasta ahora, los personajes con sus animaciones se encontraban en un formato propio del editor de Unity, de extensión .asset que generaba automáticamente la utilidad VAT a partir de los ficheros originales. Se creaba un fichero *asset* por cada animación, a partir de una malla principal y sus *point cache* asociados proporcionados por los diseñadores.

Lo que se explica de forma resumida en este apartado, es la forma de como a partir de los ficheros originales, crear un único fichero de extesión .fbx por cada personaje, conteniendo todas las animaciones juntas. Esto se realizará mediante la herramienta de software libre y código abierto Blender, en su versión 3.3.

Lo primero que hay que saber, es que esta opción es posible gracias a que el propio motor de Unity es capaz de interpretar e importar directamente este tipo de ficheros.

Esto permite, como ya se ha explicado en este documento, crear ficheros de menor tamaño y con todas las animaciones integradas, así como la posibilidad de aplicar diferentes versiones de malla posteriormente (LODs) y todo ello eliminando la dependencia de la utilidad VAT.

Las animaciones se consiguen disponiendo primero de una malla principal, llamada *pose* y posteriormente creando BlendShapes que son interpretados por el *complemento Skinned Mesh Renderer* de Unity. Estos BlendShapes almacenan los cambios de posición de cada vértice entre instantes de tiempo, los cuales de definen mediante lo que se llama un *KeyFrame* asociado a cada instante. Como la creación de estos BlendShapes se creará manualmente desde la herramienta Blender, es importante saber que, en esta herramienta, la nomenclatura usada es *ShapeKeys*.

Los pasos para seguir son los siguientes:

1. Cargar el fichero original que contiene la imagen *pose*, junto con su definición de *point cache*. Esto en Blender se hace a través del modificar *MeshCache*, mediante el cual se podrá utilizar el fichero MDD, que contiene la animación de tela procedente de la aplicación Marvelous Designer. De esta forma, ya tenemos el personaje y su animación funcionando.

2. Se exportan todos los *KeyFrames* de la animación al formato *Wavefront(*.obj)*. Esto creará un archivo de extensión obj por cada KeyFrame, siendo el primero de todos el que usaremos como malla principal o *pose*. En la exportación, deberemos de seleccionar las opciones "Export Animation" y seleccionar el rango de frames elegidos y marcar además las opciones "*Grouping/Object Groups*", "*Grouping/Material Groups*", "*Grouping/Smooth Groups*", "*Geometry/Materials*.

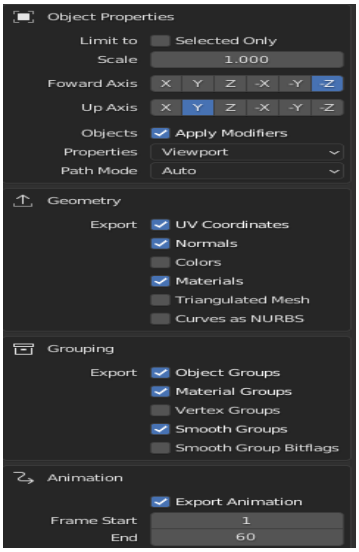


***Ilustración 21**. Exportación a obj.*

3. Ahora, en un nuevo fichero de Blender, se importa el obj elegido como *pose*. Esta pose, ya no tiene armadura ni esqueleto, es simplemente una malla con materiales y texturas.

4. **IMPORTANTE:** En las propiedades del objeto  , en el apartado ShapeKeys, debemos de pinchar en **+**, lo que creará la ShapeKeys Base.

5. Ahora se importarán el resto de los ficheros obj. Como se va a crear una nueva ShapeKeys por cada nueva malla, se puede optimizar el proceso, no cargándolas todas, ya que, al provenir de una animación basada en esqueleto, normalmente tienen demasiadas *Keyframes*, algo que es totalmente innecesario en la animación por vértices que estamos creando.

   En este proyecto, por ejemplo, se utilizó una de cada diez mallas, sin verse afectada la animación y con el consiguiente ahorro en el tamaño final del fichero. Así por ejemplo, en el caso del personaje *"MujerHumildeCaminando"*, de las 60 mallas(keyframes) que se obtienen al exportar su animación a *obj*, solo se utilizaron 6.

6. Se seleccionan las mallas cargadas y como última opción seleccionamos la malla *pose,* en la cual, dentro de sus propiedades de objeto, en el apartado *shapekeys,* utilizamos la opción *"Join as Shapes",* tal y como se muestra en la figura 19. Tras esto, ya se pueden eliminar de Blender todas las mallas cargadas salvo la *pose,* que ahora contiene las *ShapeKyes*.

7. Ya solo queda, crear los *keyframes,* asociados a cada *shapekeys*.

Lo que se explica a continuación, es muy similar a lo que se haría en otras aplicaciones de diseño para crear animaciones basadas en vértices, a las también se las suele llamar *MorphsTargets*.

Es importante seguir esta receta, a modo de algoritmo para evitar que aparezcan deformaciones no deseadas.

Resumidamente, de forma sucesiva, en un *frame* en concreto, su shapekeys asociada valdrá uno (se supone que tiene el mayor peso) y la shapekeys asociada al frame siguiente y al anterior, deberán valer cero.

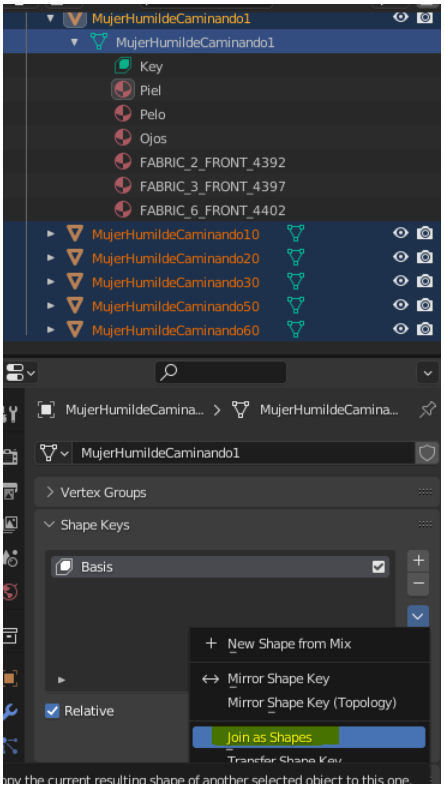


***Ilustración 22**. Creación de ShapeKeys*

Mas detalladamente, en el animador de Blender, nos colocamos en el *frame* asociado al nombre de la shapekeys, esto nos servirá de guía para no perdernos. Por ejemplo, en el caso de la shapekeys con nombre "*MujerHumideCaminando10",* nos situaremos en el *frame* número 10 del editor.

Como es la primera, el valor de esta shapekeys y de la siguiente, (en este caso la número 20) será de cero.

Nos colocamos ahora en el *frame* número 20, y en este caso, el valor asociado a la *shapekeys* 20("*MujerHumideCaminando20")*, será de 1. La shapekeys siguiente, la número 30, en este *frame* en el que estamos actualmente, que es el número 20, tendrá valor 0 ya que va a comenzar a activarse y la animación aún no tiene ningún peso.

El *frame* anterior, del que venimos, en este caso el número 10, también tendrá valor 0, ya que en este *frame* número 20, el 10 ya no tiene ningún peso.

Y así, sucesivamente.

En la ilustración 24, se puede ver el comportamiento esperado, en este caso en la animación de la *"MujerHumildeHablando"*. Se ve claramente como situados en el *frame* número 29, los valores de las *shapekeys* asociados a los *KeyFrames* número 21 y 41, varían su peso, dependiendo de la cercanía a uno u otro.

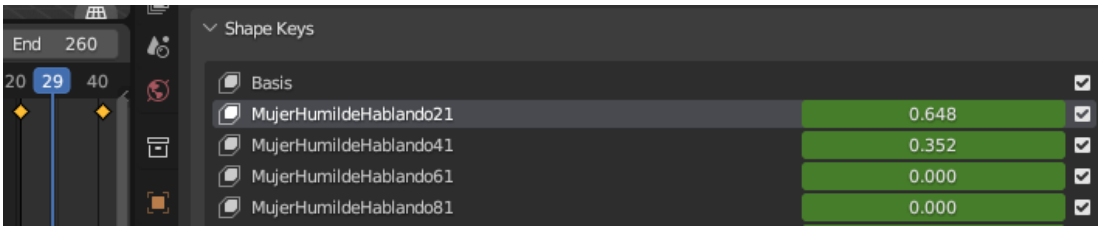


***Ilustración 23**. Cambio de pesos en shapekeys.*

8. Como última anotación, hay que indicar que para unificar todas las animaciones en un mismo fichero, hay agregar las *shapekeys* necesarias y crearles *keyframes* con el comienzo y fin en las posiciones que se decidan. Después desde el editor de Unity, se puede elegir el primer y último frame de cada animación.

***Ilustración 24**. Animaciones mujer hablando y caminando juntas en Blender.*

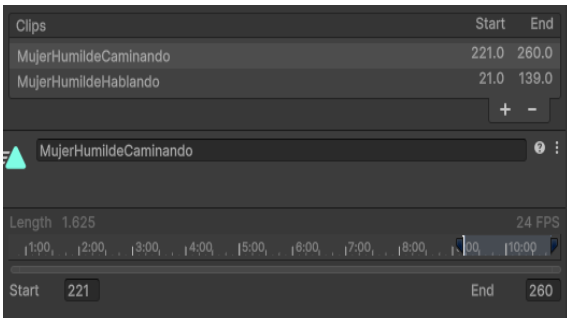


***Ilustración 25**. Animaciones importadas en Unity acotadas por número de keyframe.*

Anexo II. LOD. Retopología

Para la aplicación LOD, se necesitan varias mallas de diferentes calidades por cada personaje. Ya se explicó en este trabajo, concretamente en el punto 2.5, las dificultades de crear esas nuevas mallas aplicando retopología para reducir el número de vértices en objetos animados por esqueleto. Recordemos que la asignación hueso/vértice en este tipo de animaciones quedaría rota y habría que rehacerla. Además, la animación por vértices de la ropa no está exenta de dificultades, al tener que mantener el mismo número de vértices la malla principal o *pose* que la de sus *point cache* asociados. Entonces,

¿Cómo podemos solucionarlo sin rehacer todas las geometrías y animaciones por cada nueva malla?

Pues la solución, es realmente sencilla. Lo que se hará, es retopologizar todo junto, y con esto me refiero a cuerpo, tela y shapekeys a la vez.

Recordemos que, en definitiva, los ficheros generados en el Anexo I, solo contienen una única geometría con diferentes *shapekeys* asociados, no hay ni componentes de armadura, esqueleto, modificadores de ropa ni nada parecido, simplemente, una malla con sus materiales y texturas junto a sus *shapekeys* correspondientes.

Pues bien, partiendo de estos ficheros, y nuevamente gracias a Blender, vamos a utilizar la opción *Decimate (en este contexto sería algo así como diezmar)* que lo que hará será retopologizar la geometría incluyendo las *shapekeys* asociadas.

El procedimiento se desglosa así:

1. Se duplica el personaje en Blender tantas veces como niveles de LOD se quiera obtener.

   "Un truco para que cuando sea importado desde Unity, este reconozca automáticamente los niveles de LOD existentes, es renombrar cada objeto agregándole el sufijo _LODXX. "

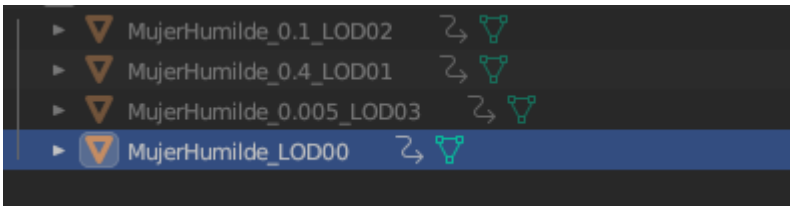


***Ilustración 26**. Niveles de LOD en Blender.*

2. Tras esto, seleccionando cada objeto de forma independiente, pasamos al modo edición desde el menú o pulsando la tecla *tabulación*, marcamos la malla completa presionando la tecla *'A'*, y desde el menú *Mesh*, vamos al submenú "*Clean Up*" y desde aquí a *"Decimate Geometry"*. La secuencia completa se puede apreciar en la ilustración 28.

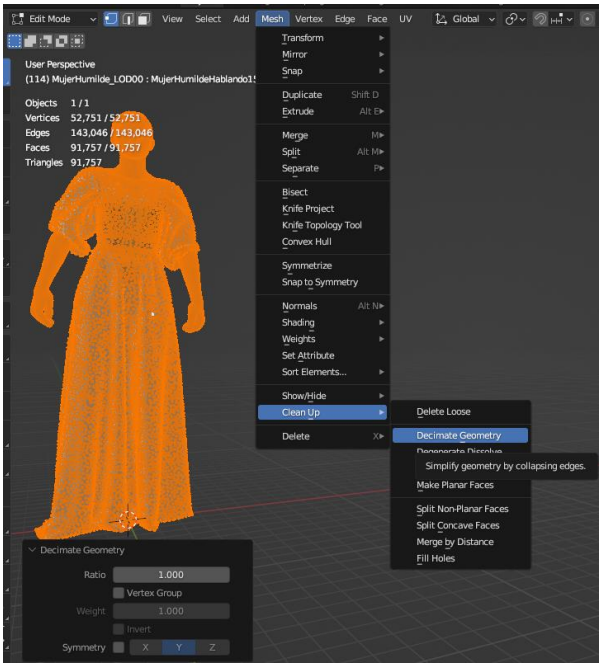


***Ilustración 27**. Camino para retopologizar malla.*

3. En el valor para *"Ratio"* de las opciones propuestas por *"Decimate Geometry"*, se establece el porcentaje, entre valores de 0 hasta 1, de la cantidad de vértices que se quiere conservar.

   En la ilustración 29, se muestra el resultado obteniendo una malla con tan solo 10% de los vértices de su geometría original. Vemos, comparado con la imagen de la ilustración 28, como se han convertido los 52.751 vértices a tan solo 8.301. Además, se ha mantenido la animación creada mediante *shapekeys*.

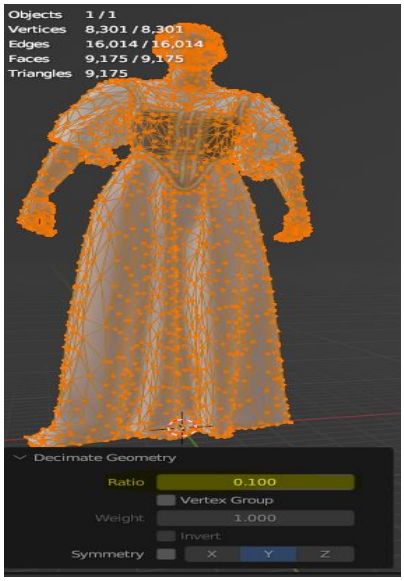


***Ilustración 28**. Retopologizar al 10% de vértices*

Portfolio : https://juansiverio.rmpixel.es/