UNIVERSIDAD EUROPEA DE MADRID

FACULTAD DE DISEÑO Y TECNOLOGÍAS CREATIVAS

GRADO EN ANIMACIÓN

PROYECTO FIN DE GRADO

Elaboración de Assets Hard Surface para Portales de Venta Online

HERNÁN ALONSO DÍAZ

Dirigido por

M.ª Socorro Pascual Nicolás

CURSO 2024-2025

TÍTULO: ELABORACIÓN DE ASSETS HARD SURFACE PARA PORTALES DE VENTA ONLINE

AUTOR: HERNÁN ALONSO DÍAZ

TITULACIÓN: GRADO EN DISEÑO DE VIDEOJUEGOS

DIRECTOR/ES DEL PROYECTO: M.ª SOCORRO PASCUAL NICOLÁS

FECHA: JUNIO de 2025

RESUMEN

Este proyecto tiene como objetivo principal el desarrollo de un entorno 3D para videojuegos que sea comercializable y funcional según estándares profesionales actuales, optimizado para su integración en Unity 6 y Unreal Engine 5. La creación de este asset ha sido el resultado de un proceso investigativo centrado en identificar los flujos de trabajo más eficientes para artistas 3D independientes, con especial atención al uso de herramientas accesibles como Blender, complementadas puntualmente con software de pago de amplia disponibilidad como el de Adobe.

Como resultado de esta investigación, se ha generado también una guía paso a paso que documenta todo el proceso seguido, con el fin de facilitar su replicación por parte de artistas con conocimientos básicos que deseen introducirse en la venta de contenido digital.

Además del desarrollo práctico, el proyecto incluye un análisis comparativo entre flujos de trabajo tradicionales y nuevos enfoques, señalando en qué contextos resulta más adecuado utilizar técnicas como el texturizado con atlas, materiales tileables o bakes únicos. También se exploran nuevas ayudas, como automatismos o la IA generativa aplicadas al modelado y al texturizado.

Palabras clave: *hard surface, props, assets de escenario, modelado 3D, texturizado, assets para juegos AAA.*

ABSTRACT

This project's main objective is the development of a 3D environment for video games that is both market-ready and fully functional according to current professional standards, optimized for integration in Unity 6 and Unreal Engine 5. The creation of this asset has been the result of a research-driven process focused on identifying the most efficient workflows for independent 3D artists, with special attention given to the use of free open-source tools such as Blender, complemented when necessary by widely available paid software like Adobe's.

As an outcome of this research, a step-by-step guide has also been produced, documenting the entire process in detail, with the aim of making it easier to replicate for artists with a basic technical background who intend to enter the digital asset market.

In addition to the practical development, the project includes a comparative analysis between traditional workflows and more recent approaches, highlighting in which contexts it is more appropriate to use techniques such as atlas texturing, tileable materials, or unique bakes. It also explores new shortcuts, such as automation tools or generative AI applied to modeling and texturing.

Keywords: *hard surface, props, environment assets, 3D modeling, texturing, AAA game assets.*

Dedicatoria

A mi hermana, mayor por dejarme jugar con sus consolas antiguas cuando era pequeño. Fue la manera de que un niño nacido en el 99 pudiera disfrutar de los grandes clásicos del videojuego.

A mis padres, por apoyar un cambio de carrera y comunidad autónoma, así de la nada, cuando ya iba a empezar tercero de carrera en otro ámbito.

A mi amigo David, por mantenerme con hambre de aprender sobre desarrollo y empujarme siempre a proyectos sobre los que no tenía ni idea a lo largo de toda la carrera, aun cuando elegir un objetivo ya conocido y que hubiera reportado unas mejores calificaciones hubiera sido mucho mejor sobre el papel.

A mis profesores, por no bajar demasiado a la tierra a un alumno kamikaze que siempre escogía proyectos que se le iban de alcance.

Por último, a la comunidad de desarrollo de videojuegos y arte 3D, que aunque por lo general solo sueltan prenda de cosas básicas o intermedias, y a veces parezca que les quemen con agua bendita cual demonio si publican papers avanzados, siempre hay algún generoso iluminado que termina explicándolo en su blog; aunque sea solo para ganar una discusión.

Luego, quisiera añadir unos agradecimientos menores. A mis gafas de filtro azul, que me permiten seguir viendo al menos a 360p, a la máquina de espalda baja del gym por corregirme la postura para seguir siendo antropomórfico e impedir que transicione por completo a programador, y a un servidor (a mí no, al de Drive, que me deja escribir gratis).

TABLA RESUMEN

	DATOS
Nombre y apellidos:	Hernán Alonso Díaz
Título del proyecto:	Elaboración de Assets Hard Surface para Portales de Venta Online
Directores del proyecto:	M.ª Socorro Pascual Nicolás
El proyecto se ha realizado en colaboración de una empresa o a petición de una empresa:	NO
El proyecto ha implementado un producto:	SÍ
El proyecto ha consistido en el desarrollo de una investigación o innovación:	NO
Objetivo general del proyecto:	Estudiar los procesos óptimos para elaborar props y escenografía inorgánicos de cara a su posterior comercialización en portales de venta online.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. INTRODUCCIÓN

1.1 Contexto y justificación

El objetivo del proyecto es arrojar luz sobre los métodos y procesos más óptimos para crear assets[1] hard surface[2] de cara a su posterior publicación en portales de venta online. Se busca averiguar qué programas son los más rápidos para cumplir qué funciones, así como desmentir mitos y acelerar el flujo de trabajo evitando todo aquello que no sea estrictamente necesario. Para conseguirlo se va desarrollar un producto como ejemplo práctico.

Buscar el flujo de trabajo más óptimo para crear assets de escenario optimizados para videojuegos permitirá acelerar el proceso y evitar redundancias. Es posible que tras una serie de iteraciones prácticas se descubra que algunos de los procesos más populares quizá no sean necesarios dadas ciertas circunstancias, o que la IA directamente los remplace.

1.2 Planteamiento del problema

Con el avance del hardware la cantidad de polígonos y el tamaño de las texturas que son aceptables dentro de un juego comercializable es cada vez mayor. El avance ha sido tan rápido que en ocasiones genera una dicotomía, y es que hay artistas que continúan optimizando al máximo sus productos aún cuando se podrían ahorrar gran cantidad de trabajo si no lo hicieran tanto, y luego, en el extremo contrario del espectro, artistas que confían tanto en las nuevas capacidades del hardware que directamente entregan productos que en muchas ocasiones serían totalmente inaprovechables en un caso de uso real.

Especialmente en el mundo de los assets orientados a "render"[3] de interiores para el ámbito arquitectónico e inmobiliario comienza a ser algo habitual encontrarse con modelos con cantidades abusivas de polígonos, densidades geométricas muy poco equilibradas entre las diferentes zonas del modelo y repletas de "ngons"[4]. ¿Es esto fruto de las prisas y del ansia por generar "loops de refuerzo"[5] a mansalva para poder pasar a "subdivisión"[6] lo más rápido posible o se trata de que para su caso concreto de uso ya no es necesario modelar "bien"? ¿Es un modelo "bueno" cuando sigue las directrices adecuadas de optimización o cuando cumple estrictamente con el mínimo trabajo viable para el uso que se le va a dar? ¿Podría ser que la nueva potencia de hardware lo permita y que ya no sea necesario optimizar los assets siempre y cuando no se vayan a usar en zonas abiertas y grandes? ¿Puede la IA hacerlo sola?

Otro punto de vista a tener en cuenta para la optimización del flujo de trabajo es cómo se va a comercializar. Si los modelos van a emplearse de forma nativa en un único proyecto, es decir, se están desarrollando exactamente aquellos que se van a usar en el juego, tendría todo

[1] **Assets 3D:** elementos visuales que componen una escena digital. Pueden ser modelos, animaciones, efectos, partículas, etc.

[2] **Hard Surface:** modelado 3D de objetos rígidos y mecánicos como armas, armaduras, edificios o vehículos.

[3] **Render:** imagen final generada por ordenador a partir de un modelo 3D, con luces, texturas y efectos.

[4] **Ngon**: polígono de 'N' caras. Normalmente polígonos de más de 4 caras, cosa que puede generar errores de visualización.

[5] **Loop de refuerzo**: línea extra de polígonos que endurece bordes al suavizar un modelo. Ayuda a mantener la forma al subdividir.

[6] **Subdivisión**: aumentar el número de polígonos automáticamente para hacerlo más suave y detallado.

el sentido condensar las texturas de los diferentes modelos en el mínimo número de imágenes y que varios modelos compartan una misma (atlas). En cambio, si los assets van orientados a venderse a terceros en un portal de venta online y no están diseñados específicamente para un proyecto, sino que más bien son genéricos, es posible que el cliente solo quiera incorporar algunos de modelos al proyecto y que no le interesen todos. En este caso, ¿sería recomendable condensar las texturas en un atlas? ¿No sería mejor ir a textura por modelo para no obligar al cliente a mantener texturas absurdamente grandes que corresponden a modelos que no va a utilizar ocupando espacio en su carpeta de proyecto? Si en cambio sí que se van a emplear la mayoría, ¿el hecho de no condersalas no aumentaría el tamaño en disco?

El presente proyecto tiene como objetivo, además de desarrollar un producto comercializable, investigar acerca de cuáles son las tendencias de mercado en portales de venta para poner solución a estas cuestiones. Todo esto manteniendo el trabajo del artista 3D al mínimo realmente requerido.

1.3 Objetivos del proyecto

1.3.1 Objetivo General

Investigar acerca de los flujos de trabajo más óptimos para diseñar, modelar y texturizar un paquete de assets 3D arquitectónicos y de props[7] para videojuegos en un entorno de hardware moderno y desde el punto de vista del modelador autónomo. Así mismo, también se pretende estudiar las formas más eficientes para integrarlo en motores gráficos e indagar sobre su posible venta en portales web.

1.3.2 Objetivos Específicos

1. Estudiar las formas más eficientes de diseñar un paquete de assets 3D, así como modelar sus componentes, desde piezas grandes que sirven de referencia general del escenario hasta piezas pequeñas o “clutter” encargadas de decorar. Después, analizar las formas rápidas de texturizar todos los elementos e integrar el paquete en motores gráficos, preparado para su venta en portales web.
2. Marcar como mínimo viable de cara al proyecto final la investigación acerca de la integración y preparación a estándar profesional del producto en los motores gráficos[8] Unity y Unreal Engine 5. Establecer como lugares de venta objetivo sus respectivos portales habituales, que a fecha de diciembre de 2024 son “Unity Asset Store” y “Fab”.

[7] **Prop**: objetos interactivos o decorativos. Complementan la escena en la que se encuentran, son elementos secundarios. Son un subtipo de asset 3D.
[8] **Motor gráfico:** programa central donde se construyen y ejecutan los videojuegos integrando varias funciones esenciales como renderizado, físicas, animación o gestión de la lógica del juego. Aquí se aúnan todos los componentes (modelos 3D, programación, VFX, SFX, etc).

3. Tratar de definir baremos claros para acotar qué flujos de trabajo son los óptimos en función del tipo y la necesidad de cada modelo. Asignar qué programas usar para cada proceso por velocidad y eficiencia.
4. Esclarecer un flujo centrado en la viabilidad económica para autónomos, con un foco principal en software gratuito y con una gran comunidad de creadores para compartir información, como es el caso de Blender. Buscar integrar al máximo este approach con los motores modernos y tratar de relegar las licencias de pago a casos de uso específicos.
5. Indagar mediante la práctica acerca de la viabilidad de automatizar procesos como el marcado de costuras y el despliegue de UVs[9].
6. Indagar mediante la práctica acerca de cuándo es posible en assets para videojuegos pasar por alto partes del flujo de trabajo que se creen muy arraigadas. Entre ellas crear versiones "high"[10] de los modelos de cara a transferir información de iluminación, y el bakeado de mapas.
7. Esclarecer qué papel puede tener la IA en la creación de assets a fecha de 2025.
8. Establecer formas de adaptar el paquete de modelos a las necesidades de un proyecto en marcha, es decir, que puedan funcionar de manera autónoma si es que algunos assets no hacen falta y se eliminan (las carpetas de proyecto pesan mucho).
9. Buscar cómo ahorrar espacio mediante el uso independiente de los canales del 'RGBA'[11] para asociar mapas de texturizado a cada uno de ellos y, por tanto, condensar varios mapas en una sola textura (métodos "MRAO" en Unreal y Metallic + Smoothness en Unity).
10. Lidiar con la preparación de assets para importación en Unity, ya que puede ser engorroso. Su gestión de materiales y texturas es totalmente diferente al estándar si no se cuenta con plugins externos o "Render Pipelines" auxiliares. Unity, por convenio, trabaja con algunos mapas invertidos y con otros combinados, con estándar PBR Metallic o Unity PBR.
11. Esclarecer cómo integrar assets hechos en programas de nueva democratización en Unreal Engine, ya que está pensado para funcionar con el flujo tradicional de normales y mapas en estándar PBR MRAO de Autodesk.
12. Averiguar cuáles son las estructuras, nomenclaturas y convenios de cara a organizar el paquete a estándares de industria, de forma que pueda ser un producto en venta en las tiendas web principales de cada uno de los motores gráficos citados.
13. Investigar cómo vender el paquete en los portales de venta, tanto temas contractuales con las webs como posicionamiento de producto.

[9] **UV:** sistema de coordenadas bidimensional mediante el que las partes de un modelo 3D se asocian a una imagen 2D. Las UVs determinan dónde y cómo se ensambla una imagen para que esta se renderice sobre el modelo.
[10] **High Poly:** en este contexto, versión alternativa y súper detallada de un modelo que se usa para transferir información a su versión menos detallada. Es esta última la que se renderiza en el juego por motivos de rendimiento, pero ahora cuenta con el detalle transferido.
[11] **RGBA** (Red, Green, Blue y Alpha): sistema mediante el cual los píxeles representan los colores y la transparencia.

Capítulo 2. ANTECEDENTES

2.1 Contexto

La venta de assets 3D a través de plataformas digitales se ha consolidado como un modelo de distribución habitual tanto para desarrolladores independientes como para estudios consolidados. Espacios como Unity Asset Store o Fab de Epic Games permiten comercializar modelos reutilizables para videojuegos, realidad virtual, visualización arquitectónica o simuladores, abriendo así una nueva vía profesional para artistas técnicos y generalistas.

Este ecosistema comercial, sin embargo, presenta características muy particulares: el contenido debe ser lo suficientemente genérico como para ser útil en distintos proyectos, pero a la vez lo bastante estilizado o técnico como para destacar frente a la competencia. A esto se suman criterios de optimización, compatibilidad entre motores y claridad en la organización del material, que no siempre coinciden con los flujos clásicos del desarrollo 3D para un proyecto cerrado.

El contexto tecnológico actual, por otro lado, permite asumir mayores cargas geométricas y resoluciones de textura que en generaciones anteriores. Gracias a herramientas gratuitas como Blender y a motores cada vez más eficientes como Unreal Engine 5 y Unity 6, el acceso a producción de alta calidad se ha democratizado, permitiendo a individuos desarrollar paquetes completos con aspiraciones comerciales sin necesidad de infraestructuras corporativas (Benzo, 2024).

Este proyecto se sitúa dentro de ese entorno mixto entre lo técnico y lo comercial, con el objetivo de desarrollar un conjunto de assets arquitectónicos y utilitarios de estilo medieval compatibles con Unity y Unreal Engine. El enfoque no es solo crear un producto final utilizable, sino también identificar qué buenas prácticas siguen siendo esenciales, cuáles pueden flexibilizarse, y cómo equilibrar calidad visual, limpieza técnica y viabilidad comercial en un mercado en constante evolución.

2.2.1. Perspectiva Histórica del Ámbito

Para comprender adecuadamente el punto de partida desde el que se plantea este proyecto, resulta necesario hacer una breve revisión de cómo ha evolucionado el desarrollo de gráficos 3D a lo largo del tiempo. Lejos de tratarse de un simple recorrido cronológico, esta mirada retrospectiva permite entender cómo se han ido asentando ciertos flujos de trabajo, por qué persisten algunas convenciones técnicas, y en qué momentos concretos se introdujeron herramientas o metodologías que hoy en día se dan por sentadas. Solo a partir de

esa evolución es posible valorar con criterio qué procesos siguen siendo relevantes y cuáles pueden replantearse o eliminarse sin comprometer la calidad del resultado.

Aunque existen ejemplos puntuales anteriores (trabajos doctorales y aplicaciones militares), el mundo del 3D empieza a existir como tal en los años setenta, que es cuando primero se publican juegos con gráficos tridimensionales básicos. Videojuegos como "Maze War" (1.er juego en 1ª pers.) o "Spasim", ambos en 1974, ya introducen entornos wireframe[12] rudimentarios (DigiBarn, 2004).

La llegada de las computadoras personales en los 80s como el 'Amiga' o el 'Commodore 64' permite comercializar juegos con mejores gráficos 3D al contar ya con hardware dedicado. Aunque "Battlezone" de Atari (1980) ya empieza a abandonar el 'wireframe' al presentar colores planos sobre los polígonos, y Zaxxon y Q*bert implementan la proyección axonométrica, no es hasta 1984, con el juego "Elite", que ya se presentan entornos 3D más complejos, incluyendo simulación de profundidad y los precursores de las texturas mapeadas[13] actuales (Donovan, 2010).

Es en los años noventa llega una mejora grande en el hardware gráfico, con tarjetas como las "Voodoo", que permite la democratización de los juegos 3D. "Wolfenstein 3D" (1992) populariza las texturas mapeadas, y "Quake" (1996) ya incorpora el mapeado complejo de texturas tal y como se conoce hoy en día. A partir del año 1994-1995 el 3D pasa a ser el nuevo estándar con la salida de las consolas PlayStation y Sega Saturn (Kent, 2001).

A primeros de la década del 2000, tal y cómo dicta el segundo volumen histórico de Kent (2021), es gracias al hardware de Sony (PlayStation) y a su buen soporte a desarrolladores en nuevos motores gráficos como Unreal Engine y CryEngine que se desbloquea el modelado de alta resolución para videojuegos, así como mejoras drásticas en iluminación. A lo largo de esta década se consigue pasar de entornos 3D sencillos a entornos que representan la realidad. Aparecen técnicas de texturizado para conseguir mayor detalle sin falta de aumentar la carga poligonal de los modelos como el 'mapeado de normales'. Se acelera la creación de personajes con la aparición del esculpido digital gracias a ZBrush (año 2000) y el polypaint, que permite pintar directamente sobre el modelo y sin requerir de UVs (actualización de ZBrush 3.0, 2007). Los mapas de UVs se vuelven más complejos y las texturas de alta resolución son el nuevo estándar (Kent, 2021).

En los 2010s, se consigue alcanzar entornos 3D fotorealistas gracias a la democratización del uso del escaneo 3D, a la captura de movimiento, al renderizado basado en físicas (PBR), a versiones más potentes de motores gráficos, especialmente Unreal Engine 4 y

[12] Wireframe: representación visual básica de objetos 3D. Solo muestra las líneas que conforman los bordes de los polígonos que componen el modelo.
[13] Texturas mapeadas: que ya contienen información sobre cómo se han de aplicar imágenes 2D en torno a un modelo 3D, ya emplean 'UVs'.

Frostbite 3 (ZeusAl, 2013), y a nuevos flujos rápidos y realistas de trabajo de la mano de software como Substance Painter (2014) que permitían pintar en tiempo real directamente sobre el modelo con mapas PBR y mediante un sistema de capas.

A día de hoy, unos de los principales avances en 3D de cara a videojuegos son las nuevas tecnologías que va introduciendo Unreal Engine 5 en sus versiones: “Nanite” para el tratado de geometría compleja y “Lumen” para mejorar iluminación (Benzo, 2024). También cabe resaltar el avance de la IA para acelerar procesos, las nuevas tecnologías de realidad aumentada y virtual , y por último, la democratización de la industria que supone la popularización de software gratuito como “Blender” o el uso de las comunidades cooperativas de creadores para compartir conocimiento como pueden ser “Reddit” o “Youtube”.

A continuación, se recogen los autores más relevantes consultados a raíz de los cuáles se parte para la implementación del proyecto. Se desgranará qué usos y aplicaciones han tenido de cara al desarrollo del producto.

2.2 Referentes

2.2.1. Limitaciones Bibliográficas Actuales

A día de hoy, prácticamente no existe bibliografía que aborde con profundidad cómo se producen *assets* 3D ajustados a estándares modernos dentro de la industria del videojuego AAA[14]. Lo poco que hay trata aspectos básicos, o bien lo hace desde herramientas que han quedado anticuadas, o desde marcos conceptuales que no alcanzan los niveles de exigencia actuales. Tampoco abundan textos que expliquen de forma práctica cómo se relacionan entre sí las distintas etapas del proceso, y eso es algo que inevitablemente condiciona el enfoque de cualquier trabajo que pretenda analizarlo de forma completa. Y es que este vacío se debe a una combinación de factores: por un lado, el ritmo al que cambian las tecnologías y los métodos de trabajo haría poco rentable escribir libros avanzados que van a quedar desactualizados en cuestión de pocos años; y por otro lado, existe un secretismo evidente dentro de las grandes empresas del sector, tanto para proteger propiedad intelectual como para no dar ventaja a la competencia. A esto se suma una fuerte competitividad entre profesionales independientes, que habitan un contexto laboral donde, en lugar de compartir herramientas o conocimiento, a menudo prima más guardárselo.

Por todo ello, las fuentes realmente útiles para este proyecto no han sido libros, sino otros canales de información más dispersos y cambiantes. En parte, está la documentación que compañías como Epic Games o Autodesk publican para acompañar sus lanzamientos —a menudo vaga, pero lo bastante clara como para orientar ciertos pasos—. Por otro lado, estaría la observación directa de productos profesionales terminados, analizados a fondo mediante ingeniería inversa o simplemente replicando estructuras técnicas detectadas en su

[14] **AAA (Triple A):** juegp de gran presupuesto y altas expectativas comerciales. Equipo grande detrás y tiempo largo de desarrollo.

implementación. Es en este tipo de referentes, más prácticos que teóricos, donde se ha apoyado el enfoque de trabajo de esta investigación.

Otra fuente importante de información han sido los tutoriales disponibles en plataformas como YouTube, así como ciertos ciclos formativos de pago enfocados en herramientas específicas. Sin embargo, este tipo de contenidos suele presentar dos problemas fundamentales: o bien son excesivamente básicos, pensados para iniciarse desde cero, o bien abordan técnicas avanzadas de forma aislada, sin ofrecer una visión de conjunto ni explicar cómo encajan dentro de un flujo de trabajo coherente.

2.2.1. Selección de Referentes

A pesar de las limitaciones mencionadas, sí existen ciertos autores y recursos que han servido de base, aunque fuera parcialmente, para este proyecto. Algunos funcionan como punto de partida sobre el que construir; y otros, aunque estén desactualizados en ciertos aspectos, siguen siendo útiles por los fundamentos que establecen. A continuación, se recopilan los principales referentes consultados, indicando brevemente qué aporta cada uno y de qué forma concreta se ha tenido en cuenta para las decisiones adoptadas durante el desarrollo.

Referente	Contribución principal	Aplicación al proyecto
Andrew Gahan	Explica el proceso completo de creación de arte 3D para videojuegos, desde modelado hasta texturizado.	Se usa como punto de partida para comparar los flujos tradicionales con los actuales, y decidir qué pasos pueden optimizarse o replantearse.
Owen Demers	Introduce conceptos clave sobre pintura de texturas y organización del trabajo por capas.	Sirve como base para estructurar las texturas y establecer una jerarquía clara de estándares PBR.
Luke Ahearn	Amplía información sobre creación de entornos y texturizado técnico para motores de juego.	Referencia útil para contrastar cómo abordar materiales, optimización y compatibilidad en motores modernos.
David Baldwin	Se centra en diseño de entornos y props arquitectónicos para cine y videojuegos.	Aporta criterios compositivos y formales aplicables a la creación de assets coherentes con estilos realistas.

Jason Osipa	Explica cómo razonar una topología eficaz para facilitar animación y deformación.	Útil para decidir cómo resolver ciertas zonas del modelado aunque no se vayan a animar, priorizando limpieza y adaptabilidad.
John Hughes et al.	Explican gráficos 3D a nivel técnico-matemático.	Sirven como referente histórico.
Jason Gregory	Ofrece una visión completa de la arquitectura de los motores y cómo se renderizan los assets en ellos.	Ayuda a tomar decisiones sobre organización de materiales, canales técnicos y ajustes de shader desde el punto de vista del motor.

Tabla 1. *Resumen de referentes*.

Además de los autores citados, hay algunas puntualizaciones relevantes que conviene tener en cuenta:

– John Hughes et al., en *Computer Graphics: Principles and Practice*, siguen siendo una autoridad indiscutible para los fundamentos matemáticos y estructurales del 3D. Aunque el enfoque está más orientado a la programación que al desarrollo artístico, sus aportaciones permiten contextualizar muchas de las prácticas actuales.

– Los libros de Andrew Gahan, aunque centrados en software como Maya o 3ds Max, siguen funcionando como guía estructural gracias a su claridad y enfoque didáctico. Algunos aspectos técnicos han quedado obsoletos, pero los principios generales que recoge siguen siendo útiles.

– En el área del texturizado, Owen Demers se centra más en la lógica visual y la organización conceptual del proceso, mientras que Luke Ahearn aporta una visión práctica, aplicada directamente a motores y juegos reales. Ambos se complementan bien.

– David Baldwin trabaja desde un enfoque más cinematográfico, pero lo hace con criterios formales que se ajustan a la fidelidad visual exigida en el desarrollo de entornos para videojuegos. Sus planteamientos se pueden extrapolar.

– Aunque Jason Osipa escribe para animación facial, su razonamiento sobre cómo diseñar topologías limpias y con sentido estructural ha resultado útil incluso en geometría estática, puesto que una silueta que disminuye progresivamente soportando los ejes y respetando la

dirección natural, aunque menos optimizada, ayuda a eliminar algunos errores de iluminación que pueden aparecen en caras grandes coplanares.

– Finalmente, Jason Gregory, con *Game Engine Architecture*, ofrece el tipo de conocimiento que rara vez se encuentra en libros de arte 3D: una explicación muy avanzada y completa del funcionamiento interno de los motores gráficos. Esto ha permitido tomar decisiones mejor fundamentadas en lo relativo a shaders, materiales y organización técnica de los assets.

Estas fuentes, aunque en muchos casos no respondan de forma directa a las preguntas que plantea este proyecto, sí han permitido establecer una base crítica desde la que examinar qué partes del flujo tradicional conviene conservar, adaptar o sustituir.

Ahora bien, para entender de forma precisa qué decisiones se han tomado durante el desarrollo del paquete de assets y por qué se consideran justificadas desde el punto de vista técnico, es necesario aclarar previamente algunos conceptos fundamentales. El siguiente apartado expone el marco conceptual sobre el que se apoya este trabajo: las definiciones, convenciones y principios operativos que estructuran la lógica interna del proyecto.

2.3 Marco Conceptual

Introducción

El propósito del proyecto es conseguir assets adecuados para estándares modernos de la manera más rápida posible, por lo que se estudiarán muchos de los tópicos más asentados para comprobar si a día de hoy realmente es importante seguirlos al pie de la letra o si no suponen un problema para el nuevo hardware. También a qué tipo de asset aplicaría cada uno de los diferentes flujos de trabajo.

Este apartado expone el contexto inicial que motivó la investigación y dio origen al desarrollo del proyecto. La fundamentación teórica aquí recogida proporciona las bases necesarias para comprender tanto las motivaciones como el punto de partida. Cabe recalcar que el apartado 3, dedicado a la metodología, amplía esta perspectiva incorporando los marcos teóricos que fueron emergiendo a medida que se avanzaba en el proceso de desarrollo.

2.3.1. Modelado

En modelado hay dos temas principales que pueden retrasar el flujo de trabajo. Son "crear versiones high poly" para transferir detalle y por otro lado, la necesidad de "coser geometría".

Geometría Flotante vs Geometría Cosida

Para evitar errores de renderizado, los vértices que conforman los polígonos de un modelo deben estar distribuidos como, o poder ser reducidos, a polígonos de 3 o 4 lados, no más. Esto se aplica a todos los vértices consecutivos, unidos directamente, es decir, los que están cosidos. Si por el contrario existe geometría colindante visualmente, pero que no está unida entre sí, son piezas separadas, no se aplica. Esto tiene su inconveniente, que se verá más adelante.

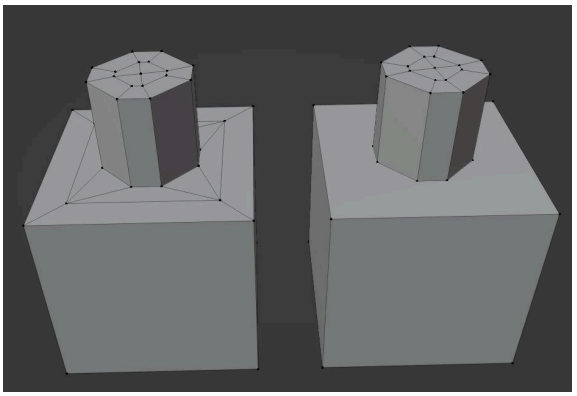


Figura 1. *Piezas cosidas (izquierda) frente a geometría flotante (derecha).*

Cuando la geometría está "cosida", cada uno de los vértices conecta con otro creando una única malla sólida, sin recovecos. Esto conlleva un mayor uso de geometría, pero todas las caras poligonales creadas están visibles y no se malgasta renderizado (Gahan, 2011). En la Fig.1, debajo de la pieza pequeña no hay nada, la geometría de la tapa del cubo va aumentando hasta acomodarse a la base del cilindro: son la misma pieza.

El caso contrario es el de la geometría flotante, que permite un modelado más rápido y de menor carga poligonal al estar separado en piezas, pero tiene la desventaja de que hay partes del modelo que se están renderizando porque técnicamente existen, pero que luego no son visibles a cámara (ver Fig.1) donde la pieza pequeña tapa al cubo se renderiza igualmente, hay una parte de la cara de la tapa del cubo que no se ve, pero está ahí debajo.

Gahan (2011) defiende que la ventaja de la reducción de la carga poligonal pesa más frente a la desventaja de la distancia renderizada inútil en la mayoría de los casos, y solo es cuando las piezas son muy grandes en unidades de tamaño cuando coser las piezas es preferible de cara al rendimiento.

Una ventaja poco conocida de la geometría flotante, según Mythmatic, es que se puede ahorrar mucho tiempo en el bakeo[15] de texturas porque solo se tiene en cuenta la proyección de sombras del "Ambient Occlusion"[16] en el eje perpendicular a la cara principal, luego la geometría puede estar completamente separada de la pieza que va a generar las mismas sombras en la textura.

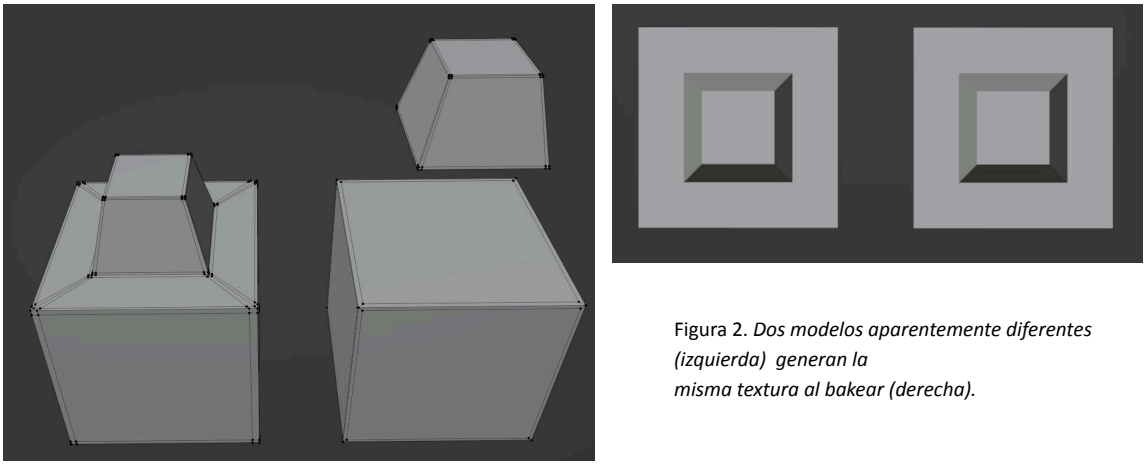


Figura 2. *Dos modelos aparentemente diferentes (izquierda) generan la misma textura al bakear (derecha).*

En la Fig.2 ambos modelos están soportados para subdividir (*ver siguiente apartado*) y generan la misma textura bakeada. El de geometría flotante, además de ser mucho más rápido de modelar, contiene tan solo 150 triángulos, frente a los 252 de la figura cosida. Esto acelera muchísimo la creación de versiones high poly de modelos complejos, cosa que se explica a continuación.

Elaboración de High Polys

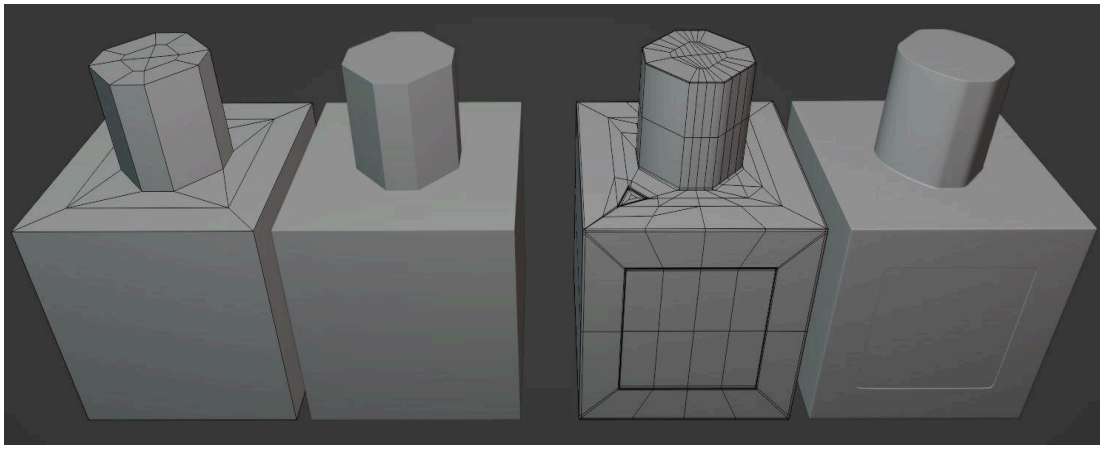


Figura.3. *Low Poly (izquierda) frente a su versión High Poly (derecha). El high poly aquí mostrado puede bakearse al low poly de la izquierda sin ningún problema para transferir detalle.*

15 **Bakear**: transferir información. En texturizado es calcular sombras y relieves, en modelado es pasar detalle del 'high poly' para usarlo en el 'low poly' sin sacrificar rendimiento.

16 **Ambient Occlusion**: sombreado que simula la oscuridad en zonas donde la luz ambiental penetra con dificultad, como esquinas o cavidades.

Según la mayoría de autores consultados, para conseguir un acabado realista es casi obligatoria la elaboración de una versión "high poly" desde la que transferir la información de iluminación (profundidad, relieve, suavidad y otros detalles), pero es un proceso que se hace cada vez más lento escalando exponencialmente con la complejidad del modelo.

Como puede verse en la Fig.2, para hacer la versión detallada es necesario crear "loops de soporte" (filas de aristas adicionales en los bordes) para mantener las proporciones al duplicarle la geometría a la pieza. Luego, se hacen aquellos detalles extra que se quieren traspasar procurando que no sobresalgan mucho con respecto a la versión original, o de lo contrario se generarán artefactos[17]. Estos nuevos detalles, requieren, otra vez, de crearles loops de soporte.

Por todo esto, en ocasiones el proceso de crear versiones high poly lleva más tiempo que modelar la pieza original, además, programas como Substance Painter o 3dsMax a día de hoy ofrecen la alternativa de utilizar la propia malla original como su versión high poly de cara a bakear los mapas de texturizado. Si bien es cierto que no se pueden añadir detalles nuevos de esta manera, se consigue el mismo suavizado final y se ahorra por completo el proceso, a parte de no tener que lidiar con los problemas consecuentes del bakeado de highs, que muchas veces genera artefactos. Esto puede ocurrir aún cuando la distribución de islas de las UVs es la correcta y las nuevas protuberancias a transferir están dentro de los límites aceptados, obligando a crear islas adicionales en lugares poco convenientes para solucionarlos. Otra manera de crear ese detalle nuevo que ofrece el high, es trabajarlo posteriormente en la textura.

Uniformidad y Optimización de las Mallas

Otro aspecto teórico a subsanar es la optimización de polígonos, y es que la mayoría de fuentes aseguran que la carga poligonal ha de reducirse al mínimo. No obstante, cuando se dejan porciones muy grandes de una pieza sin cortar porque son coplanarias[18] y técnicamente no hace falta más geometría para representarlas, cosidas a secciones con carga poligonal mayor, se pueden generar errores de iluminación en la pieza, que no cuenta con una distribución poligonal lo suficientemente uniforme. No ocurre en la mayoría de casos, pero es algo a tener en cuenta. Algunas veces estos errores es posible solucionarlos simplemente colocando loops adicionales, pero otras veces simplemente no se puede y sería necesario regenerar las caras ahora con mayor carga poligonal y mejor distribuida.

Otro motivo para dejar cierta geometría extra en lugares coplanarios donde en principio no haría falta podría ser un mayor control en la colocación de las texturas sobre las UVs (ver 3.5 Resultados).

17 **Artefactos**: errores de iluminación. Por lo general son zonas oscuras no deseadas que pueden tener bordes dentados. Normalmente se deben a problemas en la malla del modelo, a errores de UVs o a una información de texturizado que no corresponde con la de la pieza.

18 **Coplanario**: que pertenece a un mismo plano, paralelo entre sí.

También existe cierta discusión entre usuarios de calibre del subreddit 3DModelling acerca de cómo cerrar tapas. Unos aseguran una mejor carga poligonal sin efectos secundarios al colapsar las aristas al centro, y otros aseguran que esta práctica provoca errores de iluminación y es mejor cerrarlas en matriz de quads para interpolaciones más suaves y que respeten mejor la forma. Esto se discute tras elaborar el proyecto en el apartado Resultados.

2.3.2. Texturizado

Procedural vs Basado en Imágenes

Texturizar un objeto aplicando directamente imágenes finales sobre el modelo solía ser muy rápido, ya que se consigue una base fuerte muy deprisa, pero requiere de ajustar meticulosamente las caras del modelo a las texturas y de tener acceso a una librería de texturas. Además, si no se planifica bien el asset, es posible que las imágenes seleccionadas no sirvan porque no se ha medido bien el tamaño y/o la resolución y sea necesario conseguir nuevas imágenes. Esto es especialmente acusado cuando se texturiza mediante "atlas"[19] o "trim sheets"[20].

Texturizar proceduralmente[21] tiene una curva de entrada más lenta al requerir de la preparación de nodos y parámetros interrelacionados para conseguir el mismo resultado, además del conocimiento extra requerido, pero es más rápido de aplicar al modelo y es totalmente ajustable, por lo que los fallos de planificación son mucho más rápidos de subsanar.

A día de hoy, Substance Painter permite texturizar directamente casi cualquier cosa, proceduralmente y sin necesidad de preparación previa, pues ofrece materiales muy customizables ya preconcebidos para casi cualquier tarea, por lo que desaparece parte de la barrera de entrada. Además, la combinación de las técnicas procedurales junto con el bakeado de mapas permiten crear máscaras inteligentes para combinar materiales, aplicar desgastes y demás detalles con tan solo una combinación de clics, consiguiendo acabados muy realistas en poco tiempo. Ya no es necesario pintar las máscaras a mano. También permite trabajar con texturas tradicionales y aplicar alphas (Kumar, 2020). Con estas herramientas y un mínimo de práctica, texturizar es notablemente más rápido que por métodos tradicionales. Eso sí, si se busca diferenciación en el producto y que no se identifique a simple vista como "material de Substance", se requiere de un tiempo sustancialmente mayor.

Aún así, si se buscan acabados concretos o por decisión artística, el resto de flujos siguen estando vigentes, como es incluso el caso de 'World of Warcraft', donde los desarrolladores llevan más de 20 años texturizando manualmente pintando sobre las UVs.

[19] **Atlas de texturas**: dícese de una distribución según la cuál varios assets emplean una textura única en común.
[20] **Trim sheets:** distribución según la cuál varios assets hacen uso de partes comunes en una textura. Similar a un 'atlas', pero basado en la reutilización de secciones.
[21] **Texturizado Procedural**: técnica que genera texturas mediante algoritmos en lugar de usar imágenes, permitiendo patrones infinitos y ajustables sin necesidad de mapas externos.

El texturizado en base a imágenes continúa siendo el estándar en lo que a assets optimizados se refiere (Moioli, 2022). También se usa mucho en estudios con grandes librerías de texturas y escáneres propios.

UVs

Si bien elaborar mapas de UVs es un proceso metódico que lleva tiempo, a veces incluso más que el propio modelado, existen programas que permiten calcularlas automáticamente. Por ejemplo, a fecha de 2024 Substance Painter, Blender y Maya lo traen de forma integrada, a diferentes niveles. No obstante, muchos usuarios del subreddit "3Dmodeling" dicen obtener artefactos en el bakeado siempre que calculan UVs de manera automatizada. Será necesario hacer comprobaciones prácticas para dilucidar cuándo es una herramienta útil y cuándo es un estorbo.

Otro tema a debatir es el "stretching"[22], y es que varios autores alteran la distribución lógica de las UVs para evitar que se produzca. Si bien el stretching puede ser un problema, separar caras coplanares en diferentes islas de UVs también es un quebradero de cabeza, ya que provoca que haya que trabajar meticulosamente para que la transición de la textura de una a otra cara del mismo plano sea contínua y no se vea el corte. Por tanto, establecer qué cantidad de estiramiento de la textura es razonable ignorar puede ser muy valioso.

Condensar mapas de texturas

Mediante el formato "MRAO", Epic Games asegura que es posible texturizar un asset de manera realista usando solo tres imágenes: el albedo/color/diffuse (son sinónimos) por un lado, el mapa de normales por otro, y luego el "metallic", "roughness" y "ambient occlusion" condensados en uno. Esta condensación se consigue asociando cada canal del "RGB" (*red, green y blue*) a uno de los mapas, ya que estos mapas en concreto se representan en una escala de un único parámetro: cantidad de blanco con respecto a negro; utilizar tres canales para medir una única cosa es una redundancia. Será necesario investigar el flujo de trabajo para exportar los mapas en MRAO desde Substance Painter y luego incorporarlos exitosamente en los motores gráficos.

Atlas de texturas vs. Texturas individuales vs. Texturas Tileables

Los atlas de texturas consisten en una sola imagen que se aplica a varios modelos 3D por igual, eliminando la necesidad de crear todos los mapas de texturas otra vez por cada modelo individual que se tenga. Mediante UVs que apuntan a lugares reservados en la misma imagen, se establece una relación entre cada modelo y la zona que usa de la textura.

Los atlas de texturas funcionan especialmente bien para "low poly", donde puede llegar a texturizarse infinidad de modelos con una sola imagen. Al tratarse de colores planos, sirve con utilizar un único píxel por color del RGB, lo que suponiendo que se usaran todos los

[22] **Stretching**: distorsión visible de una textura causada por UVs mal distribuidos o estirados sobre la geometría del modelo.

tonos (256^3 a 24 bits) resultaría en una imagen de 16.7mb sin comprimir, llegando a ser entre 1mb y 3mb si se comprime en jpg, en función de la cantidad de compresión. Si solo se pretendiera usar unos pocos tonos por color, cosa más que probable, se podría reducir aún más el tamaño. Es una gran ventaja teniendo en cuenta que una textura realista en 4K requeriría de un mínimo de 3 imágenes si es que se usa "MRAO" y de varias más sino a una estimación de entre 6 y 12 mb por cada una; y esto por modelo, el low poly puede usar dicha imagen de 1-3 mb para texturizar un paquete entero.

En texturizado realista se complica el uso de los atlas, ya que según Moioli (2022) utilizar la misma imagen para texturizar varios modelos a resolución alta requeriría de una textura atlas de una resolución de dimensiones titánicas, y varios motores funcionan con un tope de 8K por textura. La solución habitual es evitar el atlas directamente o hacer varios atlas por temática, es decir, condensando aquellos modelos que emplean el mismo material o similar en una misma imagen. Esta segunda solución es la más utilizada cuando se analizan assets de la tienda FAB. La técnica del atlas de texturas combinada con el ya citado "MRAO" puede reducir sustancialmente el tamaño en disco de un paquete de assets sin perder en calidad, haciéndolo mucho más jugoso en la tienda.

Las texturas tileables, según Demers(2002), son un estándar para ahorrar memoria en modelos grandes. Si hacer una textura gigante ocupa mucho espacio, es más sencillo repetir una textura pequeña. Para evitar que se note dónde empieza y acaba esta repetición de la imagen existen técnicas como los "decals" o el "vertex painting", que son muy útiles de conocer, pero que por lo general son partes posteriores en la cadena de montaje, atañen al uso de motor (Unity, Unreal, etc). Aunque es buena idea tenerlas en cuenta desde el principio, lo que realmente interesa a quién hace el asset en la planificación primera es prever qué partes de un modelo tienen la suficiente repetición como para que texturizarlas de manera tileable sea una opción. Un asset bien optimizado siempre es preferible.

Por otra parte, en el portal de venta FAB se ve la tendencia a texturizar de manera individual los props grandes, ya que se van a ver de cerca por el jugador y cuentan con suficientes particularidades como para que sea inviable reutilizar. En cambio, también se aprecia un predilección por juntar en atlas aquellos objetos individuales que, aunque singulares, se van a ver a tamaño reducido por el jugador y, por tanto, no es un inconveniente que tengan una resolución menor al juntarse muchos en un atlas de tamaño limitado (4k, 8k).

Los paquetes de la tienda de más alta calidad vienen con una combinación de todas estas técnicas, y es que un mismo edificio puede venir con unas partes tileables mientras que otras van en atlas o en trim sheets, y luego traen props con texturas individuales y "clutter" (elementos pequeños) confeccionados en un atlas.

2.3.3. Flujo de trabajo óptimo frecuente en la actualidad

Según consenso de modeladores en *r/3Dmodeling*, un flujo óptimo en 2024 para assets realistas ponderando calidad y velocidad podría ser:

Modelado (Blender) -> UVs y normales (3dsMax/Maya) -> Retopologia (Topogun 2/Blender addon 'Retopoflow') -> Bakeado de mapas (Marmoset Toolset/Substance Painter) -> Texturizado(Substance Painter) -> Renderizado (Unreal Engine 5)

Este flujo es el estándar para props individuales con una textura singular asociada, pero puede cambiar drásticamente si contiene partes tileables o en atlas. Esto se estudia en el capítulo 3 de este proyecto.

Capítulo 3. DESARROLLO DEL PROYECTO

3.1 Planificación del proyecto

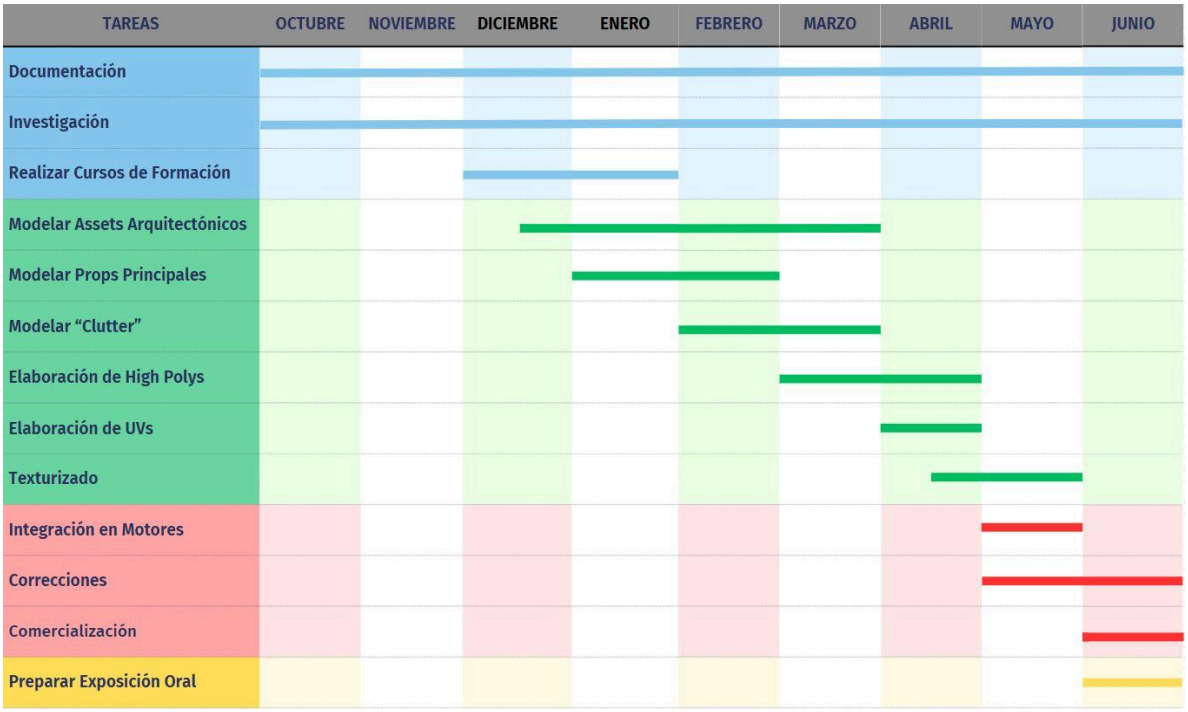


Figura 4. *Diagrama de Gantt del desarrollo estimado del proyecto. 'Azul' corresponde a preproducción y documentación, 'verde' a producción, y 'rojo' a post-producción.*

La planificación inicial de procesos (ha cambiado una vez esclarecidos los resultados) es la siguiente: se emplea Blender como programa principal y como referencia para la documentación aquí recogida por ser considerado como el programa 3D con mayor número de información compartida entre usuarios, lo que facilita la investigación. Se usa Blender sobre todo para el modelado y la elaboración de UVs. Se emplea Substance Painter para el texturizado procedural, aunque se usa Blender para texturas tileables. 3DsMax se utilizará para solucionar problemas de agrupación de normales en grupos, ya que es más versátil en esto que Blender, y Marmoset Toolbag podría emplearse de cara a transferir texturas a modelos con diferentes UVs para permitir compaginar el uso de Substance Painter (procedural y por tanto requiere de UVs 0-1[23], sin overlap[24]) con texturas en atlas (usan overlap como regla para reutilizar).

[23] **UVs 0-1**: espacio estándar de mapeado UV que abarca del 0 al 1 en los ejes U y V, dentro del cual se asignan las coordenadas para una única textura. Una textura tileable no tendría problema en salirse de este espacio. Aquello que no esté en 0-1 no puede ser bakeado correctamente.

[24] **Overlap**: coincidencia de distintas zonas UV en el mismo espacio de textura, lo que provoca que compartan la misma información visual.

3.2 Descripción de la solución, metodologías y herramientas empleadas

Las metodologías descritas en este apartado son las que a priori se creían necesarias para obtener los resultados y las que se han seguido para desarrollar el proyecto. No necesariamente coinciden con las metodologías más óptimas para alcanzar el resultado. En el apartado de *Resultados* se discuten las metodologías óptimas descubiertas una vez obtenido el producto si es que estas difieren de las descritas aquí. Esta sección no solo es una lista de instrucciones, sino que también expone un nuevo margen teórico de conceptos que han ido surgiendo a la hora de desarrollar el producto.

3.2.1 Estudio de Mercado

Lo primero para desarrollar un asset 3D para su posterior venta es decidir qué tipo de asset se va a realizar. Para esto es importante hacer un análisis de tendencias previamente para asegurar que el tiempo y los recursos invertidos en el desarrollo del producto no van a caer en saco roto.

Si bien los datos acerca de qué assets se venden más en los paquetes de venta online no son públicos y, por tanto, es difícil acotar con seguridad qué estilo artístico y temática de entornos 3D son los más populares, sí que se puede hacer una categorización aproximada fiable en base a qué contenidos se promueven más en los portales de venta. Estas webs listan productos por “más vendidos”.

Una forma rápida de esclarecer las temáticas y ambientaciones más vendidas en entornos 3D sería observar los assets más populares en cada portal y contabilizarlas. Por supuesto, se excluyen del estudio todos aquellos que no sean escenarios o props, por lo que assets exclusivos de personajes y assets de programación o animación quedan fuera. Para contabilizarlos mejor se agruparán por las categorías que se creen más probable a priori. Por temática: low poly, estilizado y realista. Por ambientación: urbano, naturaleza, medieval, ciencia ficción, guerra y otros.

	Low Poly	**Estilizado**	**Realista**
Descripción	Representación de la realidad con lo mínimo. Estilo poligonal con aristas duras, de colores prácticamente planos. Muy poco costoso de renderizar.	Representación de la realidad con algo de abstracción. Estilo de dibujo animado. Texturas pequeñas y medias, pintadas a mano. Algo costoso de renderizar.	Representación fidedigna de la realidad, a veces incluso son escáneres 3D. Varios mapas de texturas por asset, a veces pesadas. Más costoso de renderizar.
Ejemplos	Personaje de 150 triángulos.	World of Warcraft o Spyro.	Fotogrametría, asset fotorrealista.
Costes	Bajos costes en modelado, costes muy bajos en texturizado, coste bajo en optimizado.	Costes moderados en modelado, costes muy altos en texturizado, coste bajo en optimizado.	Costes medios o altos en modelado, costes medios en texturizado, coste muy alto en optimizado.

Tabla 2. *Tipos de asset por estilo visual.*

Urbano	Naturaleza	Medieval	Ciencia-Ficción	Guerra	Otros
Entornos en su estado normal inalterado de la realidad "real" y en el presente temporal.	Entornos naturales.	Entornos ambientados en la Edad Media o de fantasía.	Entornos futuristas y espaciales.	Entornos del presente, pero destruidos, o escenas postapocalípticas. Armas modernas y material militar.	No encaja en ninguno de los anteriores.
Ej: rascacielos, props de cocina, coches, props de hospital, muebles actuales, ropa moderna.	Ej: árboles, rocas, agua, flores, hierba, nieve.	Ej: tabernas de época, biblioteca de un mago, espadas, mazmorras.	Ej: ciudad del futuro, naves espaciales, entorno alienígena.	Ej: paquete de armas automáticas, rascacielos en ruinas, vehículos militares, entornos industriales abandonados	Ej: entorno del lejano oeste, paquetes para hacer blocking.

Tabla 3. *Ambientaciones generales de assets en portales de venta.*

3.2.2 Preparación Inicial

Según los resultados obtenidos (véase el apartado 3.5 Resultados) se decide hacer un asset de taberna medieval, tanto el edificio como los props asociados. Lo primero será crear una escena nueva en Blender para empezar la primera parte del trabajo: el modelado.

Se crea un cubo y se escala para que tenga el tamaño de humano promedio para tenerlo de referencia a la hora de modelar. En este caso se han elegido unas dimensiones de 1.75m de alto por 0.5m de ancho y 0.4m de profundidad.

Lo ideal para diseñar los assets es trabajar en base a referencias cruzadas, es decir, no inspirarlos en una única referencia. En Blender existen dos métodos para colocar referencias: colocar un plano en la escena al que se le asocia la referencia como textura o añadir directamente una "referencia" desde el menú de creación de formas de Blender. Las referencias están en el submenú de "imágenes" o "image" y consisten en imágenes que no tienen geometría ni se van a renderizar como el resto de texturas, simplemente están de fondo y no afectan a la escena. Esto último no estaba disponible hasta actualizaciones relativamente recientes. Es importante tener en cuenta que las "reference" de Blender no se mantienen fijas al rotar la escena, por lo que es posible que se prefiera ajustar algunas opciones en el panel de propiedades de la imagen (abajo a la derecha) para que se vean solo en vista ortográfica y no molesten para visualizar la escena en 3D.

Para modelar sobre una referencia es recomendable ponerse en vista ortográfica y usar el modo de visualización "wireframe", que permite ver a través de la pieza y seleccionar dos vértices que están en el mismo lugar aunque uno esté delante del otro. De esta manera se puede controlar con precisión dónde colocar los vértices y hacer que coincidan a la perfección con la referencia.

3.2.3 **Modelado de Piezas Grandes:** Edificios y "Hero Props"

El primer asset grande en ser modelado es la taberna. En un principio se hace en una sola pieza para tener una idea conjunta de tamaños y proporciones. Se parte de un cubo básico al que rápidamente se le divide en dos mitades en el eje X y en otras dos en el eje Y (profundidad en estándar OpenGl[25]) para posteriormente eliminar una de cada y añadir un modificador "mirror" marcando el eje X y el Y para trabajar solo en una de las mitades y que se transfieran automáticamente las acciones a las otras tres.

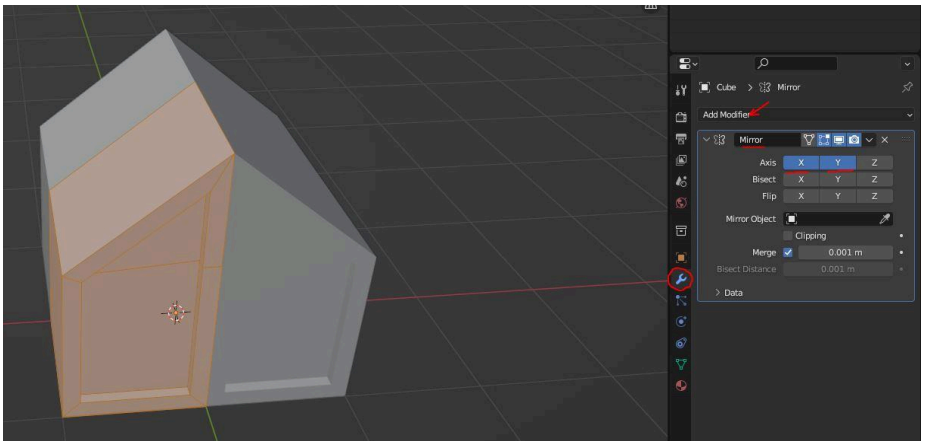


Figura 5. *Modificador Mirror en Blender.*

Se van añadiendo cortes o "loops" al modelo con Ctrl+R para tener una guía para las extrusiones pertinentes que van a ir dando forma al modelo. En el caso de la taberna, se van extruyendo hacia dentro y hacia fuera las caras necesarias para tener las primeras vigas bien diferenciadas de lo que va a ser la pared de adobe. A base de ir escalando y colocando puntos y aristas se va perfilando más la silueta del modelo final.

En base a lo que apunta Andrew Gahan en 3*dsMax Modelling for Games* se va generando la pieza entera con todas sus partes cosidas para que no haya grandes espacios de renderizado malgastado que no es visible, ya que se trata de una pieza que se va a representar a gran tamaño en el motor. Entonces, no sería presuntamente buena idea dejarla descosida aunque esto suponga una mayor carga poligonal y un flujo de trabajo mucho más lento. Esto también impide hacer selecciones rápidas de partes, ya que al ir todas en una sola pieza todo queda seleccionado a la vez al pulsarse la tecla L (selección local en Blender), igual que si se pulsa la tecla A (selección de todas las piezas). Esto último puede remediarse una vez se hacen los "seams" (ver más adelante), que una vez colocados permiten selecciones de partes por separado aunque estas estén cosidas.

Al ir cosidas las piezas, hay mucha geometría extra que se crea solo por culpa de piezas colindantes que tienen mayor densidad geométrica. Una buena práctica para reducirla es ir

[25] **Estándar OpenGl vs DirectX:** en OpenGl, el que usa Blender, la altura es el eje 'Z' y la profundidad el eje 'Y', mientras que en DirectX, el estándar de Unity y Unreal Engine, se asigna el eje 'Y' a la altura y el 'Z' a la profundidad. Esto hay que tenerlo en cuenta al importar mapas de normales, pues será necesario invertir el eje de la altura ('Y' en texturas) en algunos casos.

convergiendo loops en uno solo mediante el “merge” de puntos (Ctrl+M en Blender con ambos puntos añadidos a la selección, ver Fig.6). No importa que esto genere triángulos en lugar de quads ya que se trata de modelos hard surface que no van a deformarse. En cambio, si la idea es riggear posteriormente el prop, entonces sí que es preferible que la maya la conformen solo quads, pero no es el caso.

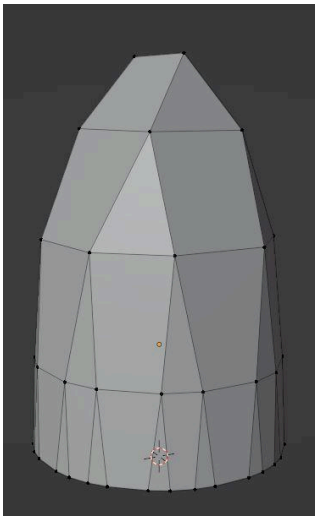


Figura 6. *Geometría que converge.*

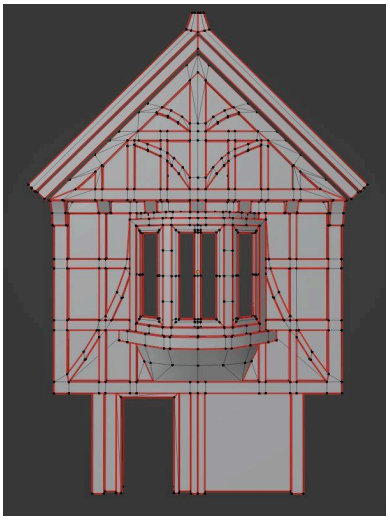


Figura 7. *Taberna en su concepción original.*

Una vez se tiene el modelo en un estado de avance suficiente en el que mantener la simetría activada con el modificador mirror ya no es una opción porque lo que queda por hacer rompe la simetría, se procede a aplicar el modificador (en el desplegable junto al nombre del modificador, botón “aplicar”).

3.2.4 **Modelado de Piezas Pequeñas:** Clutter

La ventaja frente al modelado de piezas muy grandes es que se puede recurrir a la geometría descosida, que aunque tenga una mayor carga poligonal, es un flujo de trabajo mucho más rápido. Si se separan las piezas estratégicamente, la nueva carga poligonal extra puede reducirse al mínimo, estando casi igual de optimizada o incluso más. Aunque la geometría flotante añade geometría por la inclusión de piezas nuevas, también la reduce de caras coplanarias que tenían un exceso de geometría solo para mantener la geometría consistente al estar cosidas a piezas más pequeñas.

Este flujo es más rápido porque permite añadir geometría sólo a las zonas que la van necesitando, en lugar de tener que ser consistente a lo largo de todo el modelo, lo que implicaría un trabajo manual de ir añadiendo cortes para soportar la geometría. Por ejemplo, una pared con un saliente tiene que llevar geometría extra para acomodarse a las caras del saliente si son piezas cosidas, en cambio, si es flotante, la pared puede consistir de tan solo dos triángulos.

Otra facilidad de los modelos pequeños es que no van a ocupar mucho espacio en cámara en un videojuego que los vaya a implementar: son literalmente pequeños. Entonces, requieren de menos geometría y detalle para aguantar los primeros planos. A menor tamaño, mejor pasa desapercibida una geometría menos trabajada. Si el videojuego es en tercera persona, esto se puede disimular mucho más, aunque no hay manera de saberlo si se está desarrollando un asset genérico para venta online, como se trata de este caso. Este approach de reducir detalle se aprovecha mucho para reducir carga poligonal y tener asset más optimizados, ya que si no es necesario detalle extra, directamente no se pone: mayor rendimiento para el motor.

El "clutter" se modela sobre todo teniendo en cuenta su silueta, partiendo de formas geométricas base a las que se les añade el menor número de cortes posibles. Todo aquello que no aporte a la silueta, se puede reducir. Es muy habitual ver props pequeños como copas o vasos que tienen más geometría en las aristas redondeadas o que quedan descubiertas en los ángulos de visualización más probables, y mucha menos en las zonas rectas o que es menos probable que formen la silueta. La uniformidad en la malla para mayor control del tiling mediante UVs no es tan relevante en objetos pequeños.

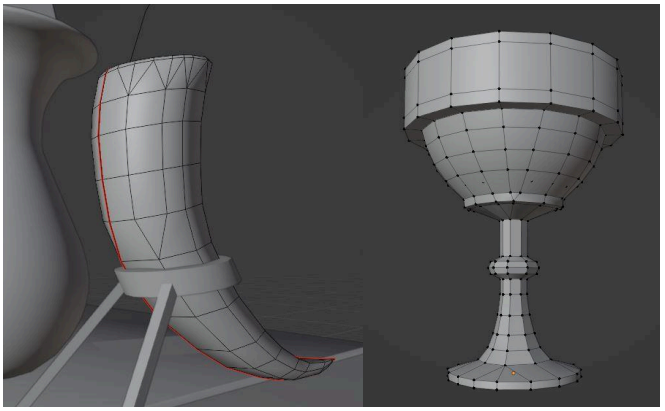


Figura 8. *Clutter que aprovecha la geometría donde la silueta es más importante y que la reduce donde lo es menos. Son resultados del proyecto, pero se exponen aquí de ejemplo.*

3.2.5 Modelado de High Polys

Todo aquel prop que vaya a ser texturizado como asset único, es decir, que vaya a constituir un material único asignado a ese único modelo, sin compartir textura con otros, va a poder beneficiarse de una versión high poly para transferirle detalle extra. Que hacer high polys merezca la pena o no en modelos hard surface se estudiará más adelante en el apartado 3.5.

Sería ideal que cada prop tuviera una versión en high, pero como la información extra de volumen y relieve que tiene el high se transfiere al "low" (original) gracias a la proyección de las sombras y luces que genera la geometría del high sobre la textura, esto impide que dicha textura sea compartida por otro modelo. La iluminación dibujada sobre la textura (bakeado) no coincidiría en otros modelos que la llevaran asignada, aparecerían sombras en lugares irracionales. Es por esto que es recomendable para materiales de carácter único.

Para realizar la versión high poly se añaden loops de soporte junto a las aristas principales que dan forma al modelo para que no se deformen posteriormente. Esto se hace de forma controlada, colocando los nuevos loops a una mayor distancia de la arista original cuanto más se quiera que se redondee esta, y a una menor distancia cuanto más dura se quiera la arista resultante. Después se le aplica al modelo una "subdivisión" gracias al modificador "Subdivision Surface" (Ctrl+2 en Blender).

Al subdividir un modelo, lo que ocurre es lo siguiente: se calcula una malla de densidad mucho mayor en base a la malla original, mediante la interpolación suavizada de los polígonos previos a su aplicación. Como curiosidad, la mayoría de software 3D subdivide mediante el algoritmo Catmull-Clark, con algunas excepciones como ZBrush, que aunque lo usa para otras cosas, usa uno propio muy diferente para subdividir.

La creación de versiones high poly es un proceso largo y muy propenso a errores de iluminación en recovecos y que es cada vez más laborioso cuanto más complejo sea el modelo, de ahí que en geometrías cosidas sea un dolor de cabeza. Por tanto, es un proceso que debe limitarse a aquellos modelos que realmente lo necesiten.

3.2.6 Creación de UVs

Aquí se discute cómo crear UVs manualmente; para ver un análisis de resultados sobre la viabilidad de UVs automáticas, véase la sección 3.5 Resultados.

La colocación de los "seams" - costuras a partir de las cuales se acota dónde y cómo se va a insertar una textura 2D en un modelo 3D - no es una ciencia cierta. Una buena guía es priorizar colocarlas en un lugar donde no llame la atención ver un corte en el patrón de la textura, y después, si es posible, colocarlas delimitando zonas coplanarias para evitar colocar a mano caras que necesiten de continuidad.

En Blender se marcan los seams de la siguiente manera: se seleccionan en el modo "selección de aristas" todas las aristas (edges) por donde se quiera crear el enganche y/o corte de la textura mediante el clic izquierdo del mouse mientras se tiene presionando la tecla Shift (ampliar selección); después, se hace clic derecho para ver el menú de contextualización de aristas y se busca la opción "mark seam". Las aristas marcadas como seam aparecen ahora en rojo. Una manera más rápida de crear la selección es seleccionar loops enteros cuando aplique. Esto se consigue añadiendo la tecla ALT a la combinación de botones ya mencionada para ampliar la selección.

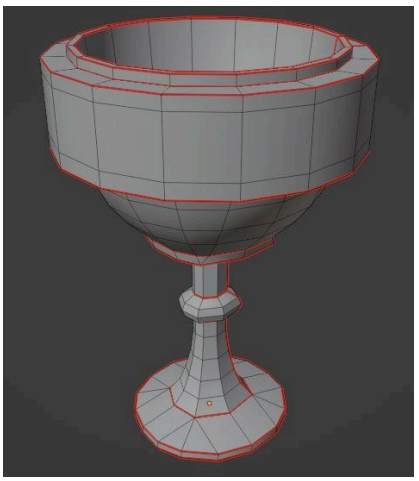


Figura 9. *Seams en un objeto.*

Una vez las aristas están marcadas como seams, se procede a desplegar las UVs (unwrap). Para ello se usa el menú UV o el atajo de teclado U y se procede a seleccionar la opción “unwrap/desplegar”. Se pueden seleccionar opciones adicionales o alternativas como “smart uv project” para conseguir distribuciones diferentes de las islas - fragmentos en los que se parte el modelo sobre la textura una vez hecho el despliegue de UVs - o para configurar el “padding” - separación en píxeles entre las islas para evitar sangrado[26] en bakeados y en la elaboración posterior de lightmaps[27] y LODs[28]. Estas opciones alternativas se exponen en la sección 3.5 Resultados. Aquí también se discute, en base a la experimentación repetida, cuándo será necesario alterar la colocación lógica de los seams por motivos técnicos.

3.2.7 **Texturizado procedural por capas:** Substance Painter

Los props grandes y principales del asset son los que deberían diferenciarlo del resto, por lo que es recomendable evitar las texturas en base a imágenes de licencia creative commons zero especialmente en este caso para evitar que se parezca al resto de assets de la tienda. Una de las mejores maneras para asegurarlo es utilizar texturas procedurales, ya que aunque se empleen materiales por defecto de Substance Painter, estos funcionan en base a ecuaciones aleatorias y nunca son exactamente iguales. Si además se usan muchas capas extra, se hacen mezclas con otros materiales y se realizan ajustes creativos personalizados, los materiales obtenidos se van a diferenciar mucho más.

PROCEDIMIENTO

Lo primero es importar el modelo en Substance dándole a “New Project”. En el menú de contextualización se presionan los tres puntos para cargar desde el ordenador el modelo. En el desplegable de la resolución se selecciona la opción que se quiera: en el caso de props pequeños, 1k o 2K, en el de props grandes, 4K, en el de props gigantes, 8K (medida muy excepcional, véase 3.5 Resultados). Después se procede a aceptar, el resto de opciones se discutirán en la sección 3.5.

Una vez importado el modelo, lo primero será bakear los mapas de normales y ambient occlusion (AO) para conseguir tridimensionalidad en el material, y el mapa de “curvature’’ para hacer máscaras. Existen más mapas que se pueden bakear como el “thickness”, pero su uso es más específico. La mayoría del comportamiento se puede conseguir con esos tres. El bakeo está escondido abajo a la derecha en “texture settings”, donde haciendo scroll se encuentra el botón “bake maps”. Una vez ahí, se deberá seleccionar la resolución del proyecto como la resolución de los mapas a bakear en el primer menú que aparece, el principal. Existen infinidad de opciones para ajustar el bakeo general y luego el de cada uno de los mapas por separado, pero estas se discutirán en el apartado 3.5 Resultados. A grosso modo,

[26] **Sangrado de texturas**: colores o detalles de una parte de la textura que se filtran visualmente hacia otra, especialmente entre los bordes de las islas UV cuando no hay suficiente espacio entre ellas. El sangrado provoca bordes oscuros, manchas o líneas incorrectas en las zonas de unión entre islas.

[27] **Lightmap**: textura que almacena iluminación precalculada para simular luces y sombras sin coste en tiempo real. Se vale de UVs.

[28] **LOD**: Level of Detail. Versión simplificada de un modelo 3D que se muestra cuando el objeto está lejos de la cámara o no necesita tanto detalle visual, con el fin de ahorrar recursos de renderizado. Las texturas en un LOD también se reducen, por lo que las islas de UVs se encogen. Es entonces cuando poco padding puede provocar sangrado.

por ahora servirá con cargar la versión high poly del modelo, si es que la hay y se pretende usarla, en la opción para ello en el menú principal de bakeo y luego darle a “bake maps”. No es necesario un high poly para bakear, es totalmente opcional. Una vez terminado el proceso de bakeo, si no da errores, ya se tiene una simulación de luces, sombra y profundidad pintada sobre la textura base que se va a multiplicar a las capas del material, así como información sobre la posición de la geometría de cara a generar máscaras.

De partida, se utiliza un material base de los que ofrece Substance, para mayor rapidez, y se construye encima. Es tan sencillo como clicar y arrastrar sobre el modelo o sobre sus UVs. Luego, se customizan los ajustes del material (colores, tiling, contrastes, etc) para conseguir diferenciación y a la vez ajustarlo a la idea que se tiene en mente. Es recomendable dejar apuntadas por escrito estas variaciones o guardar uno de los archivos de proyecto en el disco para tener más adelante una referencia exacta que ayude a dar coherencia al paquete. Así se consigue mantener cierta cohesión entre diferentes modelos del asset que usan materiales similares.

Después, se procede a jugar con texturas y diferentes materiales, con capas de color que afectan solo a hendiduras o a bordes gracias a máscaras inteligentes, etc. Para crear estas últimas, la forma más rápida es crear un nuevo “layer” (capa) en modo “fill” (rellenar) con un color plano, generalmente uno algo más claro y desaturado que el del resto del material si se quiere aplicar a los bordes, y uno más oscuro y saturado si se quiere aplicar a hendiduras. Luego, se le arrastra de la selección de máscaras inteligentes de Substance una que aplique a lo que se pretende conseguir y se suelta sobre la capa a la que se quiere que afecte. La mayoría de ellas funcionarán con tan solo tener el curvature bakeado. Ahora, ya se tiene la máscara aplicando solo el “fill layer” a los bordes o a los huecos, según se haya elegido el tipo de máscara. Es recomendable ajustar a voluntad las opciones de la máscara para conseguir el efecto deseado. Para ello, se hace click en ella en la jerarquía de capas (es el icono que aparece a la derecha del icono de la capa que no existía antes de arrastrar la máscara a la capa) y ya se podrá editar.

Si se quiere que una máscara aplique solo a una parte en concreto del material o si no se ha tenido en cuenta la orientación de las islas a la hora de crear las UVs y se quieren rotar unas sí y otras no, se puede hacer una máscara de geometría. Para ello se hace una “black mask” con clic derecho y en el modo “Polygon Fill” en modo “faces”(ver Fig.10) con la máscara seleccionada en la jerarquía de capas, se va haciendo click sobre el modelo en las caras a las que se quiere que afecte la máscara. Si no se añaden a la máscara, es necesario cambiar el color del pincel a la derecha de blanco a negro o a la inversa, ya que uno añade y el otro elimina en función de si es una “black mask” o un “white mask”. Para que esta nueva máscara aplique a una máscara inteligente que ya ha sido creada será necesario meter esta primero en una carpeta y hacer la máscara de geometría a la carpeta y no al layer, ya que solo puede haber una máscara por layer. Las máscaras de geometría, como ya se ha mencionado, también se usan para rotar islas. Por ejemplo, se puede aplicar a un material de madera para cambiar la dirección de la veta a una parte concreta. Para esto se aplica la máscara de geometría al layer, se duplica el layer, se invierte la máscara con click derecho + “invert mask” para que solo afecte

a la zona que no se tiene en el layer original, y se rota este nuevo layer desde las opciones de tiling del material. Ahora se tiene el mismo material aplicado en una dirección en un layer y afectando a una zona del modelo, y el mismo material, una vez más, en otra dirección en otro layer afectando a otra zona del modelo.

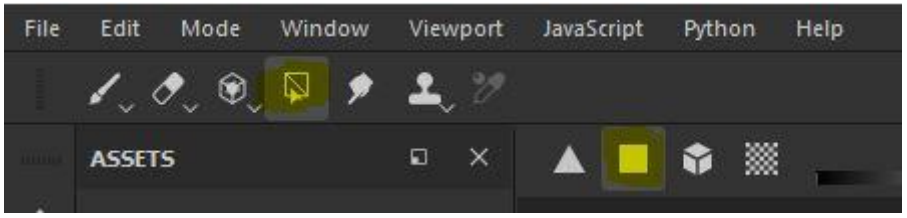


Figura 10. *Modo Polygon Fill por caras.*

Para que los nuevos fill layer que dan color a las máscaras de desgaste o suciedad no se vean planos, se puede aplicar texturas a los rellenos del layer, o directamente pintar la máscara con un "alpha".

Una vez obtenido el material deseado, se exportan las texturas teniendo cuidado del formato de exportación. Para mayor compatibilidad entre programas, es recomendable el formato PBR básico, al que se le puede eliminar el mapa "emission" (para materiales que emiten luz) si no se usa. Si es para un software en concreto, entonces es preferible usar el formato de exportación específico. Para empaquetar mapas a canales de RGB por separado, véase la sección 3.5 Resultados.

Resumen del proceso:

1. Crear un proyecto desde File-> New.
2. Ajustar la resolución del proyecto e importar el modelo.
3. Bakear mapas necesarios para trabajar con máscaras (AO, curvature, thickness, global normal) desde Texture Settings-> Bake Maps
4. Aplicar al modelo un material base o una textura para empezar a construir.
5. Añadir al menos un material secundario encima para mezclar o detallar.
6. Hacer uso de máscaras inteligentes que usan los mapas del modelo para funcionar. Permiten añadir desperfectos, suciedad y polvo de forma rápida.
7. Pintar detalles manuales encima con paint layers y con generators.
8. Aplicar detalles con alphas como texto, calcomanías o más desperfectos si es que procede.
9. Exportar las texturas con el formato de exportación PBR o Blender para mayor compatibilidad. Explorar otros formatos para situaciones específicas como motores que requieran combinación de mapas.

CONCLUSIÓN

Esta es la forma estándar de generar texturas no existentes previamente. Cuando no se quiere partir de una ya hecha (licencia creative commons zero y construir encima), crearla proceduralmente en Substance es la manera más rápida. Es posible también hacer texturas a

raíz de fotografiar o escanear el mundo real (caro y lento), de modificar imágenes de internet o hacerlas por IA generativa (ver 3.2.11).

3.2.8 Texturizado tradicional en base a imágenes: Blender

QUÉ ES Y QUÉ VENTAJAS TIENE

El texturizado consiste en asignar los diferentes polígonos de un modelo a una imagen bidimensional. Esto se hace mediante las islas de UVs, que son los pedazos 2D en los que se quedaría dividido el modelo si se recortase con tijera y se desplegara sobre una superficie plana. Una forma fácil de visualizarlo es pensar en la papiroflexia y asociar el folio plano antes de empezar a doblar, a la textura, y la figura final de papel al modelo.

El espacio que ocupa cada isla (pedazo del modelo) sobre la imagen le dice al ordenador que esa misma porción de imagen va a ser la que se proyecte sobre las caras del modelo que correspondan a dicha isla. Existen dos formas de hacer esto: colocar las islas sobre una imagen estática, o colocar una textura sobre unas UVs estáticas. Lo primero es el método tradicional y lo segundo es lo que hace el texturizado procedural (además de no requerir una imagen y crearla en el momento en base a ecuaciones). El tradicional cuenta con una imagen preconcebida sobre la que hay que colocar las islas de UVs, y ha de hacerse manualmente, es decir, el artista decide, coloca, rota y escala las islas sobre la imagen.

En la mayoría de casos sigue siendo mucho más rápido aplicar una imagen a un modelo que crearla proceduralmente porque la textura suele venir ya hecha por el material artist del equipo o directamente de una librería de texturas. Solo hay que arrastrar y rotar unas cuantas islas sobre una textura tileable y luego escalarlo todo al gusto. En cambio, si el modelo es complejo y se usan trim sheets o atlas de texturas es cuando se complica la cosa y es posible replantearse si es más rentable recurrir a la proceduralidad si la velocidad es lo único que importa. Colocar una cantidad ingente de islas a mano es una tarea laboriosa si han de coincidir en tamaño y posición con detalles concretos de la textura.

Lo bueno del tradicional es que permite “overlap”, que quiere decir que tantas islas de UVs como se quieran pueden estar colocadas en el mismo lugar de la textura. Esto da rienda suelta a la optimización, pues no hace falta tener una textura grande con muchos elementos, sino que basta con tener una textura pequeña con un único elemento que se repita por todo el modelo. También permite no respetar el espacio 0-1 de las UVs, por lo que se puede repetir muchas veces una textura pequeña sobre una superficie grande en lugar de requerirse una textura igual de grande que la superficie a abarcar. Esto último desbloquea el uso de “UDIMs” (ver 3.5 Resultados). En resumen, tiene ventajas muy grandes de cara a optimización para videojuegos, aunque sea más lento en muchos casos.

PROCEDIMIENTO

Para texturizar un modelo en Blender de forma tradicional (también permite procedural en base a nodos), lo primero es abrir dos áreas de trabajo más para dividir la interfaz en 3 ventanas. En la original y que se queda en grande, el “3D viewport” (la escena

3D), en otra y en tamaño reducido, el “Shader Editor” (donde se visualizan los nodos del material para cargar las texturas y asignarlas a sus mapas correspondientes, entre otras herramientas), y en otra y también en tamaño reducido, el “UV Editor”. Para abrir un área de trabajo se hace clic izquierdo en la esquina del 3D Viewport o de cualquier otra área de trabajo y se arrastra hacia la dirección en la que se pretenda abrir la nueva área. Por defecto se visualiza la escena, pero se puede cambiar el editor que se muestra en cada área haciendo click en el botón con el icono de una lupa que hay arriba del todo a la izquierda.

En el Shader Editor se crea un nodo del tipo “Image Texture” (Shift+A o botón “Add”) y se duplica con Shift+D tantas veces como mapas de texturizado se vayan a utilizar. Desde el botón con icono de carpeta que tienen los nodos Image Texture se cargan las texturas. Después, se hace clic y en donde los nodos pone “Color” y se arrastra y suelta sobre el nombre del mapa en cuestión que se quiera añadir en el nodo “Principled BSDF” (el shader que se crea por defecto con todos los materiales). Esto genera una unión entre los nodos y el shader. El material ya está asignado, pero aún falta decidir cómo va a afectar al modelo. Para crear un nuevo material o añadirle uno a un modelo hay que buscar “Material Properties” (icono de la bola roja con cuadros blancos en el menú de abajo a la derecha, penúltimo icono por abajo). El material seleccionado aquí es el que se va a ver reflejado en el Shader Editor.

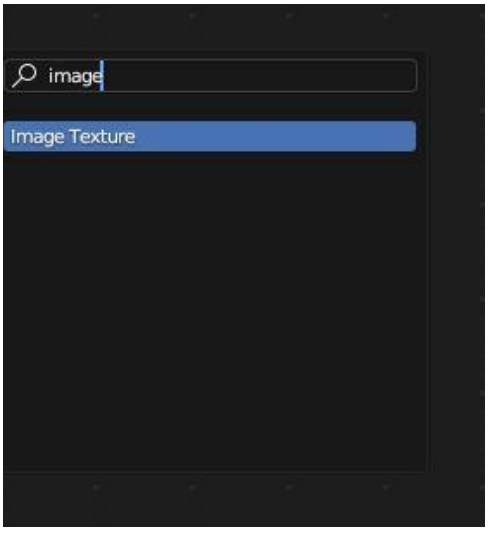


Figura 11. *Añadiendo un nodo Image Texture.*

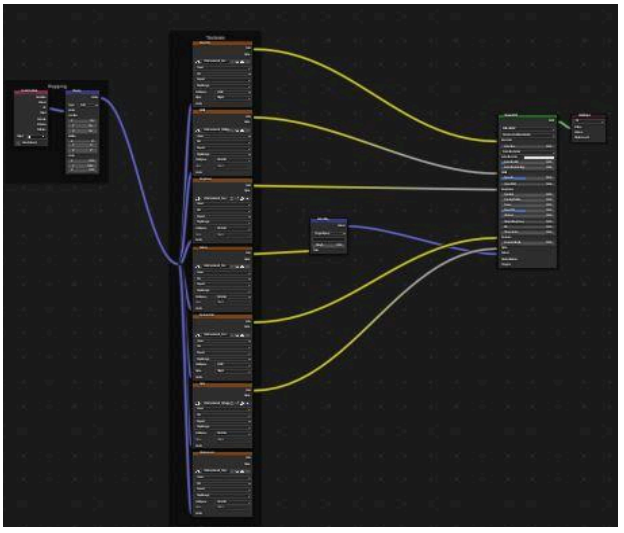


Figura 12. *Ejemplo del setup de un Material en el Shader Editor.*

Ahora, con el objeto seleccionado, se entra al modo edición (TAB). Las caras que se seleccionen van a aparecer en el UV Editor sobre el espacio 0-1. Para visualizar la textura que se quiere aplicar al modelo en el UV Editor y poder colocarle las islas a placer, será necesario buscarla en el desplegable del botón con icono de fotografía en la parte superior central del UV Editor. Si se ha cargado previamente en el Shader editor, la textura ya aparece en el desplegable. Se puede filtrar por nombre.

Se procede a seleccionar las islas enteras con la tecla L en la escena. Ahora que ya están marcados los seams (aristas rojas, ver Fig. 9) da igual que la malla esté cosida a otras piezas, que solo se va a seleccionar la geometría incluida en la misma isla. Cada isla seleccionada en la escena se ve en el UV Editor, pero esta sincronización también impide mostrar el resto de islas cuando se está seleccionando una para colocarla. Dar con el lugar idóneo para una isla es más difícil si no se pueden ver las demás. La solución híbrida es desactivar la opción "UV Sync Selection" (icono de flechas dobles en la parte superior del UV Editor). Esto permite ver sobre la textura las islas que se están seleccionando en la escena 3D, igual que antes, pero ahora no se deseleccionan de la escena por seleccionar solo una de ellas en el UV Editor. Este setup puede verse en la Fig.13.

El último ajuste necesario antes de empezar a colocar las islas es mostrar el "stretching", que es el estiramiento que tienen las caras de un modelo en el mapa UV frente a sus proporciones reales. Algo de stretching es admisible y a veces hasta inevitable, especialmente en objetos redondeados, pero mucho estiramiento destroza la visualización del modelo. Para ver el stretching hay que activar "Display Stretch" en el menú "overlays" del UV Editor (icono de dos esferas cruzadas arriba a la derecha). Ahora las islas de UVs aparecen coloreadas. Azul oscuro es cero stretching, azul claro es poco, verde ya es inadmisible en muchos casos, amarillo es excesivo, y rojo es estrepitosamente abusivo; ver debajo.

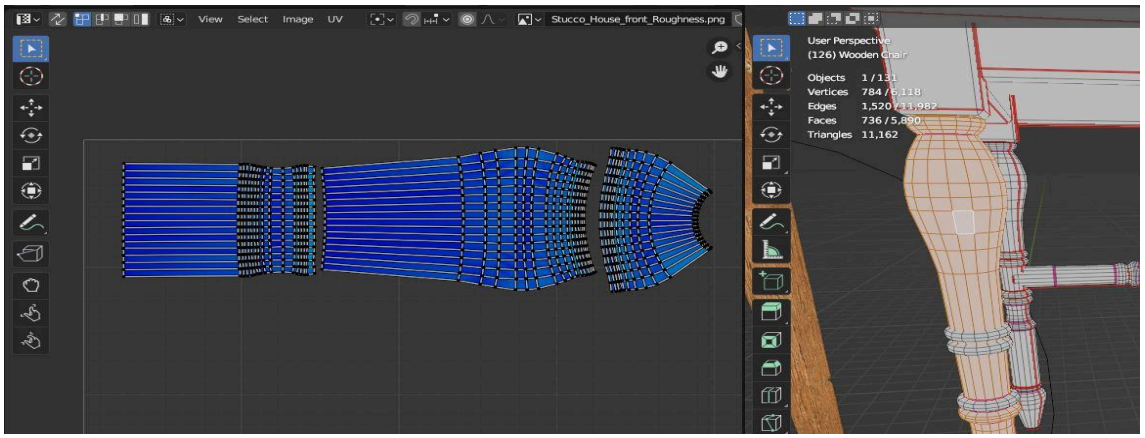


Figura 13. *UV Editor. Lo seleccionado en la escena se ve en espacio 0-1. Puede observarse cómo se genera la isla automáticamente con algo de stretching al ser un objeto redondeado.*

La forma de aplicar la textura al modelo de la manera intencionada es la siguiente. Primero se selecciona todo el modelo en escena para que aparezcan todas las islas en el UV Editor, luego se mueven todas fuera del espacio 0-1 (el cuadrado que ocupa la imagen) para asegurar que lo que se va colocando no está ocupando el espacio de nada. Luego, se deselecciona la pieza y se seleccionan ahora todas las islas de una misma índole en la escena 3D (por ejemplo, todas las vigas de madera que apuntan hacia arriba) para tener una guía visual. Luego, en el espacio UV se selecciona una sola isla del conjunto y se coloca, rota y escala para que coincida con el detalle de la imagen. Es importante recalcar que en el espacio UV no hay eje Z, por lo que el alto pasa a ser el eje Y, igual que en el estándar DirectX; luego las transformaciones son solo en la X y en la Y. Una vez se tiene colocada la isla, se procede a

colocar el resto que se habían seleccionado en la escena usando la que ya está colocada como referencia. Si no se está texturizando con overlap, será necesario seleccionar la pieza entera de vez en cuando para asegurar que no hay ninguna isla colocada compartiendo el mismo sitio que otra. Si se permite overlap, esto no importará.

Al hacerse el despliegue o unwrap de las UVs, todas las islas están a una escala (texel density) uniforme frente al modelo 3D. A mayor tamaño en el UV Editor, mayor espacio ocupan de la imagen, y por tanto, son islas que gozan de una mayor resolución. Entonces, es importante no generar descompensaciones en escala entre islas o se dará una incongruencia visual. Cuando se texturiza por overlap se sobreentiende que siempre hay algo de discrepancia en el texel density porque es imposible preparar una trim sheet o un atlas que encaje perfectamente en tamaño con todos los modelos que los puedan reutilizar, pero sigue siendo importante mantener una texel density uniforme en medida de lo posible. Hay ayudas automáticas para gestionarlo no nativas a Blender; ver 3.5 Resultados.

Proceso resumido enumerado:

1. Dividir el espacio de trabajo en al menos estas 3 secciones: 3D viewport (escena), UV Editor y Shader Editor.
2. Preparar la estructura del material para poder verlo: añadir un nodo Image Texture al Shader Editor por cada textura que lleve el material. Conectar los nodos al Shader de Blender (Principled BSDF). El albedo se deja en el modo SRGB, pero el resto de mapas se ponen en modo Non-Color o no se verán correctamente. El mapa de normales requiere de añadir un nodo Normal Map entre la textura y el Shader. Si las texturas van en formato DirectX, será necesario invertir el canal verde en el mapa de normales (ver arriba cómo).
3. Añadir el stretching a la visualización del UV Editor y desactivar la opción UV Sync Selection (ver arriba).
4. Aplicar texel density adecuado a todas las islas.
5. Colocar las islas de UVs sobre las partes deseadas de la imagen. Moverlas, rotarlas, escalarlas (con cuidado de no variar mucho el texel density).

3.2.9 **Texturizado procedural por nodos:** Blender

Obtener los resultados no hubiera sido posible de no haber aprendido manejo de nodos en Blender, es necesario para unificar estilo artístico.

Aunque para texturizar proceduralmente desde cero un material es preferible hacerlo en Substance Painter porque es más rápido, si el proyecto se monta en Blender es necesario saber manejar también en cierta medida sus herramientas procedurales. Si bien el material ya podría venir preparado desde otro software o usar de base uno ya existente de licencia creative commons zero, es posible que haga falta modificarlo para ajustarlo a la estética del proyecto. Se pueden hacer estos ajustes por separado en programas de edición de imagen, pero es más sencillo tener una visión de conjunto si se tiene todo en una misma escena bajo una iluminación común.

Estas herramientas se encuentran en el Shader Editor ya mencionado en el apartado anterior. Además de añadirse nodos para cargar las texturas, se pueden generar texturas secundarias en base a ruidos y patrones procedurales, mezclar materiales, utilizar máscaras inteligentes, hacer corrección de color, etc.

Las explicaciones pertinentes para este proyecto se van a centrar en corrección de color, aplicar máscaras y bakear las texturas resultantes para tenerlas guardadas en imagen. El resto de funcionalidades se harán en Substance Painter, pero aún así, en el anexo al documento se adjuntan recursos formativos acerca del sistema procedural de Blender para texturizar desde cero.

Para hacer corrección de color se emplea el nodo Color Ramp, que permite cambiar la coloración de los tonos oscuros, medios y claros al añadirse puntos de color. En función de la altura de la recta de valor en la que se añada el punto, afectará el cambio a una franja de luminosidad o a otra. Luego, la distancia entre los puntos influye también en el contraste entre los tonos. Para ver un funcionamiento más en detalle, ver el Anexo. Gracias a Color Ramp se puede modificar saturación, brillo, contraste, tono, e incluso añadir franjas nuevas de color en una altura de luminosidad que no existúa en la textura. Para hacerlo alterar la imagen original es necesario multiplicarla con un nodo Multiply antes de engancharla al Base Color del Shader. Con el valor numérico de Factor se regula la cantidad en la que aplica el Color Ramp o la textura, siendo 0 lo que esté conectado a la entrada A, y 1 lo que esté conectado a B.

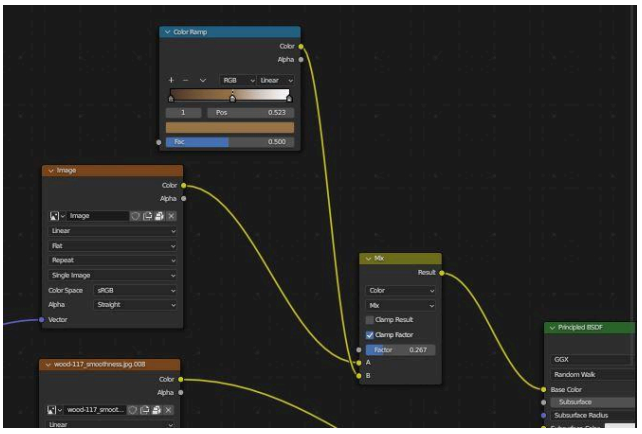


Fig.14. *Color Ramp modificando una textura en Shader Editor, Blender.*

Aplicar cambios de color sólo a una zona en concreto gracias a máscaras es el mismo procedimiento, solo que ahora el Factor en el nodo Multiply no es un número, se le asocia la textura que haga la vez de máscara. Entonces, el cambio de color se aplica exclusivamente a la zona de la textura original que coincida con la parte pintada en blanco en la máscara, y no afectará a aquello que quede en negro. Evidentemente, las texturas han de tener la misma resolución y la zona a enmascarar ha de venir preparada de antes, ya sea pintada a mano en PhotoShop o porque se haya exportado la máscara de una capa de Substance Painter (ver apartado 3.2.11). Por supuesto, se pueden combinar todas las máscaras que se quieran para poder cambiar varias partes a diferentes colores. Esto se consigue encadenando el resultado

del nodo Multiply con otro nodo del mismo tipo que ahora multiplica otro no Color Ramp con otra textura de Máscara (ver Fig.)

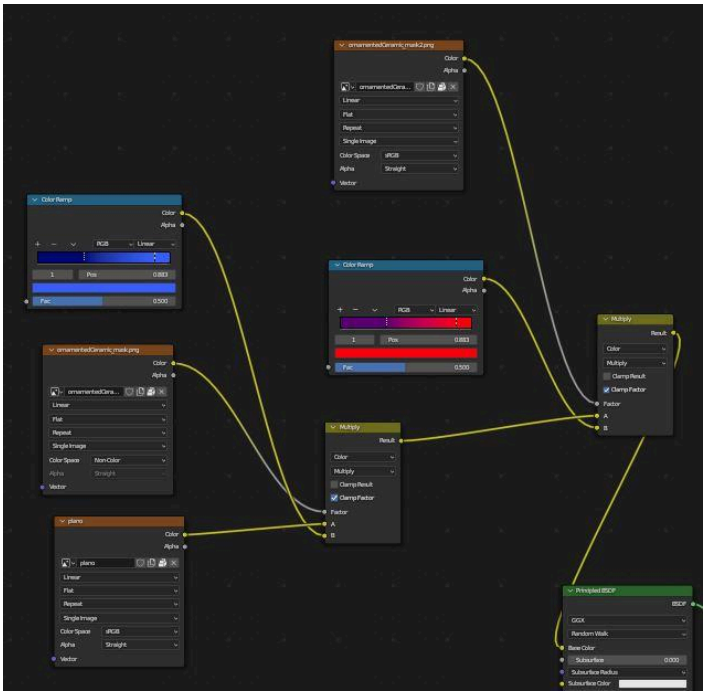


Fig.15. *Dos máscaras cambian de color partes de una textura base en Shader Editor, Blender.*

Como el resultado de aplicar nodos al Shader para cambiar el material no se ve reflejado en ninguna textura almacenable en disco de forma que pueda ser utilizada luego en otros programas, es necesario guardarlo en un imagen nueva. Esto se denomina “Bake” de texturas. El procedimiento es el siguiente:

1. Crear un plano y mantenerlo en sus proporciones originales: ha de ser un cuadrado.
2. Corroborar que las UVs del plano ocupan por completo el espacio UV 0-1 y que se trata de una única isla UV.
3. Abrir 3 áreas de trabajo en Blender a la vez: una con el 3D viewport, otra con el Image Editor o el UV editor (ambos sirven), y otra con el Shader Editor.
4. En Image Editor crear una imagen nueva desde el botón New.
5. Asignar el mismo tamaño que tiene la textura original del material a bakear a la imagen nueva. Si los tamaños no coinciden, no funcionará.
6. En Shader Editor crear un nodo Image Texture al que se le carga la nueva textura creada en el paso 4. No es necesario enganchar el nodo a nada.
7. Con el nodo creado en el paso 6 seleccionado, con el plano seleccionado en el 3D Viewport y con el Image Editor o UV Editor visualizando la imagen, ir a la ventana de Render de Blender, que es la que tiene un icono de una cámara en el menú lateral derecho (debajo del icono de una llave inglesa y encima del de una impresora).

8. Si está seleccionado el render engine Eevee, se ha de cambiar a Cycles, que es el único que permite Bake.
9. Bajar hasta las opciones de Bake. En Bake Type se selecciona el tipo de mapa que se quiere bakear (Diffuse, Normal, Roughness, Metallic, Emissive). En Influence se selecciona solo Color para que la iluminación de la escena 3D no afecte al resultado. Se marca Clear Image, y se deja View Transform en modo Standard y en Medium Contrast para evitar contrastes indeseados y resultados oscurecidos. Se puede añadir un Margin para evitar sangrado entre islas de UVs, pero como se está bailando sobre un plano que ocupa toda la UV, se deja en 0.
10. Darle al botón de Bake y esperar a que termine.
11. Desde Image Editor o UV Editor, ir a Image -> Save As para guardar la imagen en disco.

Cabe destacar que aunque las opciones de baking estén correctas y todo esté seleccionado, puede dar errores el Bake y salir una imagen negra o transparente. Esto se debe a que archivos grandes de proyecto de Blender o que encadenan varios bakeos fallidos en el mismo archivo pueden dar errores porque sí. Evitarlo es posible volviendo a guardar el proyecto como uno nuevo, es decir, guardando con Save As en lugar de con Save. También puede dar error la imagen si se visualiza desde UV Editor con el Stretching activado, es recomendable desactivarlo para la ocasión.

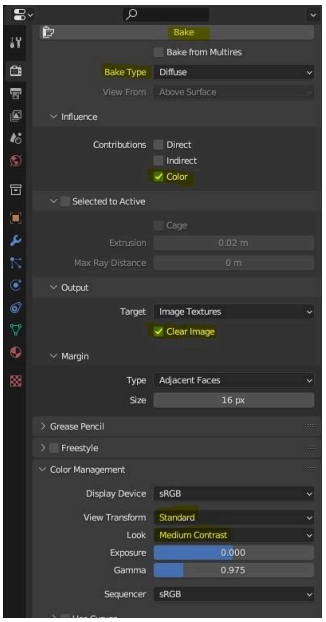
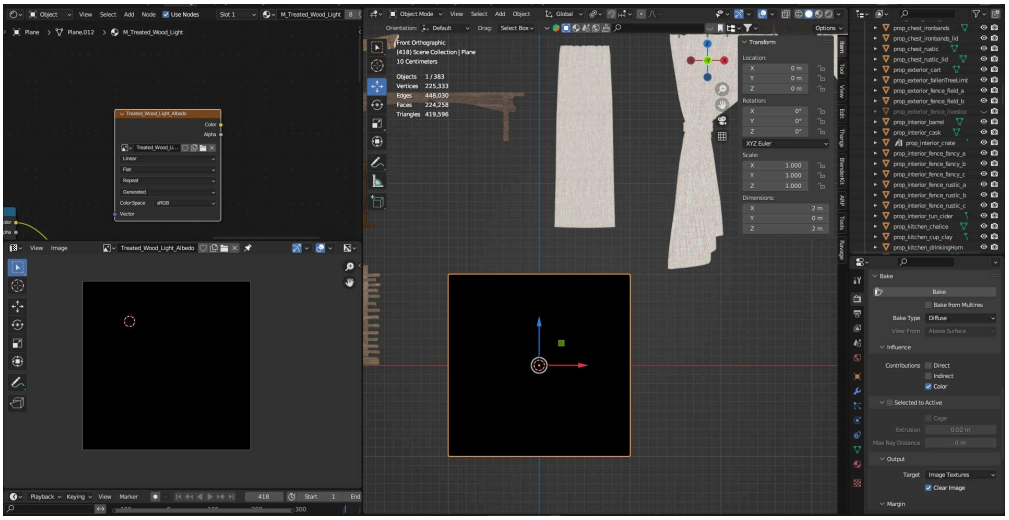


Fig.16. (Derecha) *Opciones idóneas para Bakeo en Blender.*

Fig.17. (Debajo) *Setup de Áreas de Trabajo para Bake.*



3.2.10 **Decals:** Rompiendo el tiling y falsificando ambient occlusion

QUÉ SON Y QUÉ VENTAJAS TIENEN

Una forma común de dar variedad visual a materiales tileables desde los motores gráficos es mediante el uso de *decals*. Estos funcionan como pegatinas proyectadas sobre la malla del modelo. Permiten simular elementos como grietas, desgaste, clavos, calcomanías, o incluso sombras localizadas que no existen realmente en la iluminación de la escena -simulando el efecto del ambient occlusion bakeado a la textura-. También se usan como recurso para romper la repetición evidente del tiling en superficies grandes.

A nivel técnico, un decal es una imagen con un detalle visible en el centro y un fondo transparente. A veces no se deja el fondo vacío, sino que se puede ver pintado de negro puro. Esto tiende a hacerse así porque al ser el color que se usa como máscara en el canal 'opacity', el negro puro va a ser borrado igualmente y, de esta manera, se tiene una guía visual del fondo. Aunque a simple vista parezcan imágenes opacas si tienen fondo sólido, en realidad deben contar con canal alfa para poder usar el 'opacity' después en el motor. Entonces es recomendable mantener el archivo en un formato como PNG o TGA. Usar un JPG, aunque ocupe menos, provoca bordes duros y pixelados, ya que este formato no admite una transparencia real.

Si los decals aceptan mapas de texturizado como el "normal", el "roughness" o el "metallic", ¿en qué se diferencian de un material convencional?

La clave está en su forma de aplicación: un decal no necesita UVs ni geometría extra para funcionar porque se proyecta directamente sobre las superficies dentro de un volumen definido. Es una capa visual que se suma al aspecto del objeto, sin formar parte de su estructura ni de su material base. Esto permite que múltiples props compartan un único material tileable —más eficiente y con menos draw calls[29]— mientras que los detalles individuales se aplican mediante decals.

No obstante, cada decal visible en pantalla añade su propio coste de renderizado. Usados con mesura, son increíblemente útiles para enriquecer la escena sin duplicar materiales. Pero si se abusa de ellos, sobre todo cuando se solapan en gran número, pueden multiplicar los draw calls y afectar al rendimiento. En resumen, son una herramienta poderosa si se utilizan con criterio: permiten optimizar sin sacrificar riqueza visual (Woodhouse, 2003).''

[29] **Draw Call:** instrucción de la CPU a la GPU para renderizar. Cuantos más draw calls, mayor carga en la CPU y peor rendimiento. Se reducen agrupando objetos y materiales.

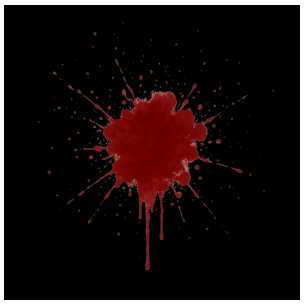


Fig.18. *Ejemplo de decal con fondo negro.*

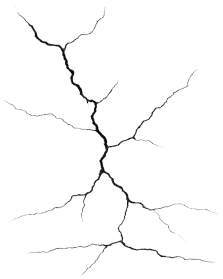


Fig.19. *Ejemplo de decal con fondo transparente.*

Aunque Blender también permite proyectar decals, al igual que ocurre con la exportación de luces o cámaras, cada software 3D tiene su propio sistema y no admite importaciones de decals ya aplicados. Esto significa que, si se quiere incluir decals en un paquete de assets para su venta o distribución, habría que entregarlos por separado, dejando al cliente final la tarea de colocarlos manualmente. Quizá haya una solución alternativa que permita un comportamiento similar sin aumentar las draw calls demasiado (véase “card decals” en la sección 3.5 Resultados).

PROCEDIMIENTO

Para crear decals, en este caso se ha usado substance Painter. Se ha exportado un plano cuadrado desde Blender para usarlo de lienzo para la textura. Si bien es cierto que un plano no permite bakear normales, no es necesario, ya que se van a usar las normales genéricas del material y no las del modelo. Se crea un fill layer en negro para el fondo con solo información en albedo, y en un layer nuevo se pinta con un “alpha” en blanco con transparencia. La biblioteca de pinceles alpha de Substance es limitada, pero si se recurre a bibliotecas de estos pinceles con licencia creative commons zero o se recurre a una IA generativa (ver más adelante), se consiguen pinceles gratis y que se pueden explotar económicamente. Mediante máscaras y cambios de color en el propio Substance, y depués en PhotoShop, se consigue el albedo final para el decal. Para conseguir el mapa de normales y otros que puedan interesar cuando no hay geometría real sobre la que calcularla se recurrirá a otro flujo posterior de trabajo (ver sección 3.5 Resultados), aunque en muchos casos el decal que se está creando se usará para detalles pequeños y ni siquiera serán necesarios otros mapas de texturizado.

Si el motor no admite el color negro como máscara para el canal opacity, se podrá seleccionar y borrar el fondo oscuro en un programa de edición de imagen como PhotoShop para tenerlo completamente transparente desde el principio.

Proceso resumido enumerado:

1. Crear un proyecto importando un plano en Substance Painter.
2. Añadir Fill Layer con solo albedo y en color negro puro.
3. Pintar con un alpha el detalle en un nuevo Paint Layer. Se pueden cargar texturas para usarlas como alphas.
4. Exportar la textura como imagen con formato que permita canal alpha (transparencia).
5. Borrar el fondo en un programa de edición de imagen si procede.
6. Usar software externo (ver 3.5 Resultados) para calcular el resto de mapas a partir del albedo.

3.2.11 Elaboración de máscaras de color: cambiando partes de forma dinámica

Una funcionalidad que suele estar altamente demandada en estudios de videojuegos que trabajan con assets externos —según la experiencia acumulada en entornos de producción— es la posibilidad de cambiar y ajustar los colores u otros mapas de texturizado en zonas concretas del modelo. Si bien los motores actuales permiten realizar cambios de color globales, estos se aplican como una multiplicación sobre el valor original en toda la superficie del objeto por igual. Como resultado, el color obtenido no corresponde exactamente con el deseado, ya que depende del color base existente, y además no puede limitarse a una región específica del modelo.

Para solucionar esto, la persona encargada del 3D o el material artist crea máscaras en las que se cambia a color blanco la parte a cambiar y se pinta de negro puro todo el resto de la imagen. Esto sigue sin solucionar que se multiplique al color original, por lo que lo ideal sería tener el asset inicialmente en color blanco puro en las partes que se crea que puede interesar personalizar el color a posteriori. Al multiplicarse con blanco, el color nuevo será exactamente el que se pretende. Esta práctica permite integrar un asset externo en la línea artística del proyecto propio al poder ajustar los colores a la paleta del juego en desarrollo, dando cohesión.

Para diferenciar positivamente un producto en venta en tienda online, incluir modelos pensados para trabajar con estas máscaras en aquellos casos en los que se considere que el cliente podría llegar a necesitarlo, es un plus grande, puesto que la vasta mayoría de assets genéricos no piensan en esto. Aunque el artista del equipo de desarrollo que integra los assets externos al proyecto termine realizando una máscara, también tendrá que rehacer la textura original para que multiplique el color con blanco; si el paquete de assets ya lo hace, eso que se ahorra el cliente.

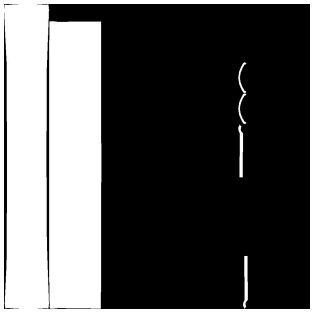


Fig.20. *Ejemplo de máscara que permite cambiar de color una parte de un modelo. En este caso se trata de un libro y la máscara afecta solo al lomo.*

PROCEDIMIENTO

Substance Painter permite exportar las máscaras como png al hacerles clic derecho del ratón. Esto exportará una imagen con la parte afectada en blanco y el resto en negro. Para generar una máscara en Substance se hace clic derecho sobre un fill layer y se selecciona la opción “Add Mask”en el menú de contexto. Luego, se puede rellenar la máscara o bien pintando a mano con la herramienta pincel, o bien usando el modo Polygon Fill (ver 3.2.7 Texturizado Procedural).

Otra forma alternativa de hacerlo, bastante más lenta pero aplicable si no se está trabajando con Substance, es hacerlo en Blender. Para ello, se crean dos materiales vacíos. A uno se le añade un nodo Emission con el color a negro puro conectado al apartado emission del Shader BSDF y al otro, un Emission en blanco puro. Se añaden ambos materiales a un modelo (menú de abajo a la derecha, la sección de materiales es el icono de bola roja, botón “+” para añadir) que tenga las UVs ya desplegadas. Ahora, se seleccionan las caras que se quieren enmascarar desde el modo edición (tecla TAB) y se le da a “Asign ” con el material blanco seleccionado en el menú de materiales que se acaba de abrir en el paso anterior (ver Fig.14). Luego, para el resto del modelo, se aplica el material negro. Ahora, en “Render Properties”, que es la pestaña con icono de cámara fotográfica en el mismo menú que el de materiales (abajo, derecha), se busca la sección Bake. Aquí se selecciona el tipo “Emit” y deja “Selected to Active” desactivado y “Clear Image” activado. Se le añade un margen de entre 2 y 4px. Se pulsa el botón “Bake”. La imagen bakeada aparece ahora en el UV Editor y en el Image Editor. Para guardarla, se hace clic en Image > Save As. Ya se tiene la máscara creada y guardada en el disco. También se pueden pintar a mano los colores blanco y negro sobre el albedo como base en un programa de edición de imagen como PhotoShop.

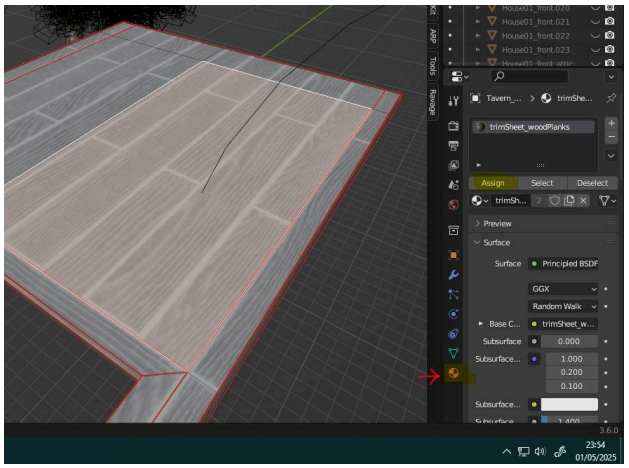


Fig.21. *Aplicar material a caras seleccionadas.*

Proceso resumido enumerado:

1. Crear dos materiales vacíos.
2. En uno, añadir un nodo Emission negro → conectarlo al shader.
3. Asegurarse de que el modelo tiene UVs desplegadas.
4. En modo edición (TAB), seleccionar las caras que se quieren enmascarar.
5. Con el material blanco activo, hacer clic en "Asignar".
6. Al resto del modelo, asignar el material negro.
7. Ir a Render Properties (icono cámara).
8. En la sección Bake seleccionar “Type: Emit”, desactivar “Selected to Active”, activar “Clear Image” y poner “Margin” entre 2 y 4 px.
9. Pulsar “Bake”.
10. Guardar la imagen desde UV/Image Editor > Image > Save As.
11. (Opcional) También se puede pintar la máscara en blanco y negro manualmente en Photoshop.

3.2.12 **UV lightmaps:** segundo canal de UVs

No es posible bakear sin errores un asset texturizado por overlap. Para hacer memoria, overlap es lo que ocurre cuando dos islas de UVs coinciden sobre el mismo espacio de la imagen. Al proyectarse iluminación sobre ellas, una tapa a la otra y le genera errores de visualizado. Normalmente, en videojuegos se suele texturizar fuertemente por overlap ya que permite mucha reutilización de texturas, y por tanto, menor tamaño en disco y mejor rendimiento en el juego (ver 3.2.8 Texturizado tradicional en base a imágenes).

Entonces, ¿cómo es posible que todos los assets optimizados para juegos no permitan el bakeado? ¿No es obligatorio prehornear el cálculo de luces en el motor para no tener que hacerlo en tiempo real en la máquina del cliente (que probablemente sea mucho peor)? La respuesta a esto sigue siendo sí, pero la clave no es evitar el overlap, sino simplemente añadir un segundo set de UVs que no lo tenga. Aquellos que modelen para Unreal Engine es posible que sean menos conscientes de esto porque el motor lo hace automáticamente, pero los de Unity sí que lo sabrán, y es que al bakear luces a assets con overlap, todo se ve raro: blanco o negro puros.

Esta vicisitud específica al ámbito de los videojuegos la permiten todos los programas generales de 3D. El convenio, si se quiere que se recojan automáticamente bien en los motores y sin cambios manuales, es que el mapa de UVs con las texturas ocupe el primer lugar o slot (UVmap 0,) y el orientado a iluminación y sin overlap ocupe el segundo (UVmap 1).

No hace falta colocar a mano en la mayoría de los casos las islas de UVs, y es que para iluminación no se requiere una continuidad entre ellas como sí que sería imprescindible para el texturizado; el resultado depende de lo que se proyecta sobre ellas, no del lugar que ocupan en espacio 0-1. Eso sí, el texel density es especialmente importante. Con todo esto en mente se han creado herramientas automáticas. Blender ofrece una muy potente, “Smart UV Project”. La mayoría de campos pueden dejarse por defecto, pero será necesario ajustar el margen (margin) en función del tamaño de la textura asociada al modelo para evitar sangrados. La documentación oficial de Unity recomienda un mínimo de 4 píxeles entre isla para asegurar que no haya errores. Según la resolución, aquí debajo se ofrece una tabla con esos cálculos ya hechos. Se recomienda redondear a la alza por seguridad, pero siempre se puede jugar con valores más pequeños para mayor fidelidad, a base de prueba y error.

Resolución de textura	**Island Margin (padding)**
256 × 256	0.0156
512 × 512	0.0078
1024 × 1024 (1K)	0.0039
2048 × 2048 (2K)	0.0020
4096 × 4096 (4K)	0.00098
8192 × 8192 (8K)	0.00049

Tabla 4. *Márgenes recomendados entre islas de UVs para lightmaps y AO.*

3.2.13 La IA para texturizado: ChatGPT y otras opciones

Para modelar, a fecha de abril de 2025, la IA más popular y supuestamente avanzada es Meshy. Se ve a simple vista en los ejemplos, que aunque consiga resultados extremadamente rápido en comparación con un humano y se centra en la parte creativa en lugar de la técnica, genera geometría desordenada y totalmente inutilizable en videojuegos, incluso en sus ejemplos de planes de pago (Heinrichs, 2024). Aún así, no es necesariamente mala geometría para escultura digital, pero eso no atañe a escala de este proyecto.

Para utilizar la IA, sirve con crearse una cuenta gratuita para tener hasta 40 generaciones gratuitas mensuales, o una limitación menor si se elige un plan de pago. Para empezar a crear se puede hacer a raíz de un texto descriptivo de lo que se pretende conseguir (prompt), o a raíz de lo más interesante y que diferencia a Meshy de otras IAs: a raíz de una imagen frontal. La diferencia grande aquí es que no escanea la imagen de referencia en busca de un "prompt" para luego generar otra imagen radicalmente nueva a partir de dicho prompt (que genera imágenes de temática similar pero sin coherencia visual con la original), sino que hace una rotoscopia a los píxeles que se le han suministrado. Esto consigue que el modelo generado tenga parecido literal con la referencia (Hu, 2024).

Otro punto favorable de Meshy a priori es su capacidad para texturizar, que parece hacerlo bastante mejor a nivel técnico que el modelado; además acepta la subida de modelos ya hechos y puede texturizarlos en diversos estilos. Tras una investigación, se razonan unos resultados más detallados en el apartado 3.5.

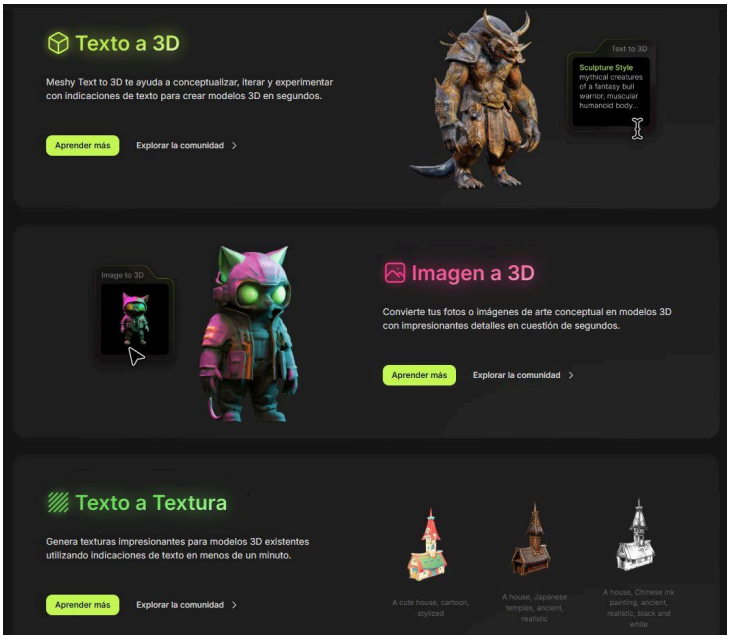


Fig.22. *Prestaciones de Meshy AI.*

El texturizado por IA parece ser más prometedor, aunque utilizar IAs generativas de imágenes directamente no es una opción viable porque entienden peor los prompts y no suelen permitir la subida de imágenes de referencia para trabajar a partir de ellas. La solución más evidente es utilizar una IA generativa de texto en conjunto con una de imágenes. Aunque existen varias a sopesar, si se reducen las opciones a que estén entrenadas a máximo nivel no solo en inglés o en chino, sino que también en castellano y que acepten imágenes de referencia como ya se ha mencionado, solo queda ChatGPT.

La versión gratuita ya permite generación de imágenes y acepta referencias del usuario en combinación con explicaciones detalladas por texto, pero está muy limitada, con un tope de en torno a 3 generaciones diarias, lo que impide trabajar a base de prueba y error hasta que se consiga lo deseado. De cara a la sección 3.5 Resultados se obtendrá el plan de pago y se estudiará en detalle cómo puede acomodarse esta IA en el flujo de trabajo del texturizado.

3.2.14 Implementación del asset en motores gráficos: Unity y Unreal

Una vez se ha terminado de producir el asset, es necesario prepararlo para su uso en motores. En función del programa de destino hay que tomar unas u otras consideraciones.

3.2.14.1 Gestión de modelos

Importar un modelo en Unity y en Unreal es muy sencillo, basta con arrastrar el archivo a la carpeta del proyecto, pero hay ciertos tecnicismos que hay que tener en cuenta; se desgranarán aquí más adelante. Ambos motores soportan la mayoría de formatos de imagen y 3D nativamente. Otros, los soportan parcialmente o mediante uso de plugins externos.

Formato	Soportado en Unity	Soportado en Unreal Engine 5	Notas
fbx	Sí	Sí	Formato estándar, ideal para animaciones y modelos.
obj	Sí	Sí	Mallas estáticas; no soporta animaciones.
blend	Sí	No	Unity lo acepta: hace conversión automática. Unreal no lo soporta (ver 3.5 Resultados).
dae	Sí	Sí	Compatible, pero puede haber errores de materiales.

3ds	Sí	No	Formato antiguo; poco recomendado.
stl	No	No	Mejor para impresión, no para juegos. Ambos motores lo importan con “plugins” externos.
gltf/ glb	Parcial (con plugins)	Sí (nativo en UE5)	Formato moderno, ideal para escenas ligeras.
ply	No (requiere conversión)	No (requiere conversión)	No se usa en pipelines de videojuegos.

Tabla 5. *Formatos 3D soportados en Unity y en Unreal.*

SUAVIZADO DE NORMALES

A pesar de que se admitan la mayoría de formatos de 3D, hay un problema con los modelos de cara a este proyecto. Se ha escogido Blender como programa principal para el modelado por la rapidez y polivalencia extra, pero se trata de un programa que no trabaja con “Smoothing Groups”, sino con su propio sistema. Autodesk (Maya, 3DsMax) sí que trabaja con ellos y permite la agrupación de normales por grupos numéricos, que es lo que se espera desde los motores y especialmente desde Unreal. Blender calcula las normales en base a los ángulos entre las caras y para controlar las normales manualmente se requiere de marcar las aristas en modo “sharp” (Blender, 2024). En otros programas esto lo controlan los grupos.

Varios artistas recomiendan ignorar el aviso de Unreal al importar mallas exportadas en Blender, pero resulta que en bastantes más ocasiones de las que se cree, este error genera problemas reales como mala visualización de suavizado de bordes: aristas duras que se vuelven blandas, artefactos, etc.

Unity, por el contrario, tienen mayor sinergía que Blender, pero a diferencia de lo que se pueda pensar, también funciona con smoothing groups. La diferencia es que si no los detecta en el objeto a importar, Unity trata de calcularlos automáticamente. Esto en la gran mayoría de los casos funciona a la perfección y el usuario no se da cuenta del problema, pero en ocasiones muy minoritarias podría llegar a haber discrepancias entre la información de normales que Unity genera y la deseada que había originalmente en Blender (Unity, 2022).

EJES Y ORIENTACIÓN DEL MODELO

Los motores gráficos, por lo general, usan el convenio +Y (arriba) y +Z (adelante). Esto ocurre así en Unity, en Godot, en Unreal, etc. También es el estándar de softwares de modelado muy extendidos, como los 3DsMax y Maya de Autodesk, pero Blender es diferente y utiliza +Z(arriba) y -Y(adelante).

Unity, por defecto detecta cuando un asset viene exportado de Blender gracias a sus metadatos y le aplica el cambio de ejes, pero casi siempre falla con el eje de profundidad por ir uno en negativo (Blender) y el otro en positivo (convenio), se genera ambigüedad. Unity entiende que ha de cambiar el eje, pero no sabe cuál es la orientación correcta de la dirección porque es algo relativo a la intención, no es globalizado. Para evitar romper animaciones, deja el eje de profundidad sin rotar. Esto, al haberse cambiado el eje de sitio, pero no rotado sobre sí mismo, provoca que quede invertido en muchos casos. Es decir, mirando hacia atrás.

Si la intención es exclusivamente emplear los modelos para un juego, no habría problemas, en realidad. Simplemente se rotan una vez importados si es que no coinciden los ejes, pero claro, si se trata de un paquete para venta, que ha de estar a un estándar profesional, no es de recibo. Ver un “-90” en un eje, por ejemplo, no es lo indicado. Además, tener los ejes sin aplicar correctamente (todo en 0) puede dar problemas a la hora de mover el asset en cuestión por código; la rotación de partida ya no es la esperada.

Un fbx puede exportarse especificando qué direcciones son las correctas, es decir, se le puede indicar qué ejes corresponden a qué ejes en la plataforma de destino y en el archivo se cambian para que coincidan con lo que se le indica. La solución obvia es indicar aquí la convención de Unity, que de hecho es como viene por defecto al exportar, pero no es una buena idea porque sigue sin solucionarse el problema de la ambigüedad. Entonces, el approach más indicado es mantener los ejes tal y cómo se usan en Blender, y después en Unity, indicarle al motor que no ha de traducir nada, que utilice los ejes que se le dan en el archivo. Esto se conoce como bakear ejes, o “bake axis” en inglés. Se desglosa cómo llevar a cabo esta técnica en 3.5 Resultados.

3.2.14.1 Gestión de texturas

Según su documentación oficial, los dos motores soportan gran variedad de texturas, pero la recomendada en videojuegos a fecha de 2025 es PNG, incluso para mapas que no usan la transparencia. Esto es así porque este tipo de archivo no lleva compresión, y por tanto, no se pierde calidad. Existen otras extensiones sin pérdidas como TGA o TIFF, pero pesan más, de ahí que sea el estándar. En base a experiencia propia, hay estudios que solo admiten PNG en sus texturas. Cabe resaltar que la lectura de PNGs puede ser más lenta y requiere de tratado especial si se pretende almacenar un mapa en su canal alfa. Entonces, es preferible usar TGA, pero solo para casos específicos donde se requiere velocidad y no se pueden precargar las texturas, o donde se necesite mayor precisión en empaquetamiento de mapas que usan canal alfa.

Formato	Soportado en Unity	Soportado en Unreal Engine 5	Notas
png	Sí	Sí	Muy usado; soporta transparencia (canal alfa).
jpg / jpeg	Sí	Sí	Ligero, pero no soporta transparencia y se degrada progresivamente en cada guardado.
tga	Sí	Sí	Usado en industria; soporta canal alfa y mantiene su información real, no solo lo usa como transparencia.
psd (Photoshop)	Sí	Sí (limitado)	Importa capas, pero no recomendado para builds.
tiff / tif	Sí	Sí	Alta calidad; archivos grandes.
bmp	Sí	Sí	Formato muy pesado; no recomendado para proyectos grandes.
gif	Sí (solo primera imagen)	No recomendado	Solo la primera imagen; no para animaciones en motores.
hdr	Sí	Sí	Usado en iluminación HDRI para entornos (skybox).
exr	Sí	Sí	Ideal para lightmaps y mapas de alta dinámica de rango.

Tabla 6. *Formatos de imagen soportados en Unity y en Unreal.*

Una ventaja de trabajar con Unreal desde Blender en contraposición a Unity es que Unreal sí admite la mayoría de mapas de texturizado tal y cómo se trabajan en Blender y otros

software estándar de 3D como los de Autodesk o Adobe. Unity, en cambio, no trabaja con esos mapas exactamente, sino que algunos los espera invertidos y otros los espera combinados en uno solo (flujo Metallic-Smoothness, ver Tabla 5). En Unreal se pueden combinar también (método MRAO), pero es opcional y el material se verá bien de primeras.

Mapa estándar	**Contraparte en Unity**	**¿Hay que invertir o modificar?**	**Notas importantes**
Albedo (Color, Diffuse)	Albedo	No	Se usa directamente como textura base de color.
Normal Map	Normal Map (OpenGL). Ver explicación más adelante en este epígrafe.	Invertir canal verde (G) si estaba en estándar DirectX.	Unity espera normales estilo OpenGL (Y+ hacia arriba).
Roughness	Smoothness (en canal alfa del Metallic)	Invertir todo el mapa (blanco ↔ negro)	Unity usa Smoothness en lugar de Roughness.
Metallic	Metallic-Smoothness (R=Metallic, G=AO, B vacío, A=Smoothness)	No	Se empaqueta junto con Smoothness en un único mapa RGBA.
Ambient Occlusion (AO)	AO Map (opcional, en input separado o en metallic smoothness)	No	Puede usarse directamente como input en shaders que lo permitan.
Height (Displacement)	Height (opcional en shader)	No	Normalmente no requiere inversión. Depende del shader que se use.
Emission	Emission Map	No	Directamente conectable en el shader Standard de Unity.

Tabla 7. *Mapas de Texturizado en Unity y cómo se interpretan.*

Para ver un setup extendido de cómo se conectan e importan estos mapas en Unity y en Unreal, ver 3.5 Resultados.

Otra sinergía de Unity con Blender es que ambos usan el estándar OpenGl, donde el eje vertical del espacio UV (bidimensional, es una imagen 2D) aumenta hacia arriba. Unreal Engine usa DirectX, donde el eje crece hacia abajo. Esto quiere decir que en OpenGl el origen de coordenadas (0,0) está en la esquina inferior izquierda y en DirectX, en la superior derecha. Esto es importante diferenciarlo porque aunque no afecte visualmente a mapas de texturizado bidimensionales, que son todos menos el de normales, sí que afecta a los que son tridimensionales. Un mapa de normales calcula la profundidad y el relieve codificando direcciones en 3D, a diferencia del "height map" que solo usa "alto" o "bajo". Entonces, al interferir tres dimensiones, el eje Y se ve invertido en un estándar frente al otro. Esto quiere decir que un mapa de normales exportado en Blender de la forma estándar se va a ver bien tras ser importado en Unity , pero en Unreal se va a ver invertido, con las hendiduras donde debería de ir el relieve y viceversa. Para solucionarlo, o bien se invierte antes de sacarlo de Blender (véase cómo en el siguiente párrafo), o bien se invierte en la ventana de importación de Unreal (hacer clic en "Flip Green Channel" en la pestaña de "Compression Settings").

Para invertir un mapa de texturizado en Blender, ya sea para conseguir un smoothness de Unity a partir de un roughness o para cambiar de estándar un mapa de normales, se necesitan tres nodos: "Separate Color", "Invert Color" y "Combine Color". Se añaden al Shader Editor con Shift+A y se colocan en el mismo orden en que se acaban de enumerar. Primero se pone el nodo Separate Color después del nodo de la textura uniendo "Color" con "Color". Luego, se coloca el Invert Color. Hacen falta tantos nodos de este tipo como colores que se quieran invertir. Se asocia a un Invert Color el color de salida del nodo Separate Color que se vaya a invertir. En el caso de un mapa de normales, solo ha de invertirse en color verde del RGB, el resto se conectan directamente al siguiente nodo (Combine Color) sin invertirlos previamente. Por último, se conecta el Combine Color, al que se le asocian todos los colores, invertidos o no. La salida del Combine color se conectará directamente al Principled BSDF, a no ser que se trate de un mapa de normales, en cuyo caso se conecta primero a un nodo Normal Map (ver debajo, Fig. 15).

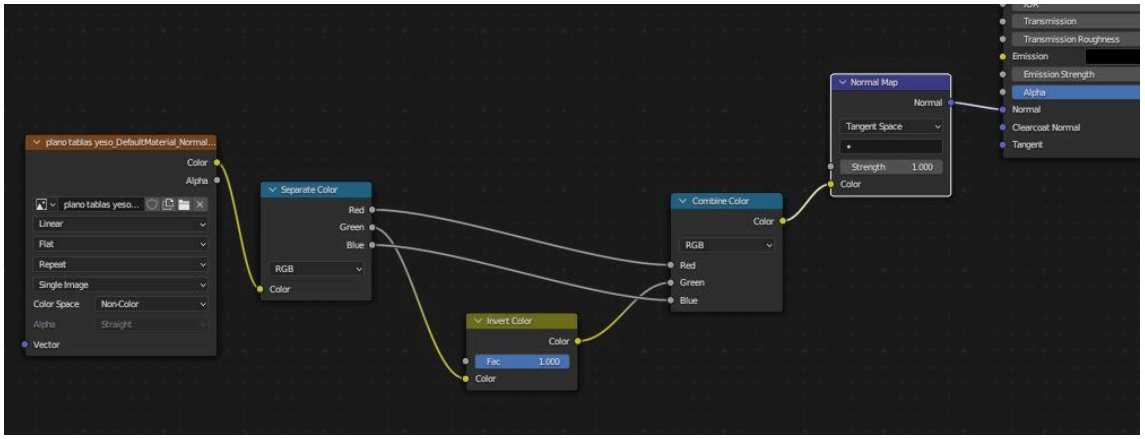


Fig.23. *Cómo cambiar de estándar OpenGl a DirectX, o a la inversa, un mapa de normales en Blender.*

Para saber cómo se exportan las texturas una vez modificadas desde Blender, es decir, no la textura base que se importa, sino la que ha sido cambiada desde el Shader Editor, véase 3.5 Resultados. También se explica ahí el flujo para combinar mapas en Unreal: método MRAO.

La organización y la nomenclatura propias de cada motor también difieren, al igual que los mapas y shaders que no se pueden exportar desde los programas de 3D, que deberán hacerse específicamente en cada motor (emission, cristales, etc). Esto también se desglosa en la sección de Resultados.

3.2.15 Puesta en venta: Unity Asset Store y Fab

Por último, una vez el asset ya está listo para su venta, se procede a distribuirlo.

Según la documentación asociada de Unity Asset Store y de Fab, para ser vendedor a fecha de abril de 2025 es necesario:

Requisito	Detalles
Cuenta de Unity	Poseer una cuenta activa.
Perfil de Publicador	Estar registrado en el Publisher Portal y haber completado la información fiscal.
Cumplimiento de Directrices	Ajustarse a las Submission Guidelines de calidad técnica y contenido.
Información Fiscal	Entregar formularios de impuestos (W-8/W-9) según país.

Tabla 8. *Requisitos para ser vendedor de Unity Asset Store.*

Requisito	Detalles
Cuenta de Epic Games	Es necesario tener una cuenta activa en Epic Games.
Registro como vendedor	Registrarse en el Publisher Portal de Fab.
Información fiscal	Entregar formularios de impuestos (W-8/W-9) según país.
Cumplimiento de directrices	Asegurarse de que los activos cumplan con las Directrices Técnicas de Fab.

Tabla 9. *Requisitos para ser vendedor de Fab.*

Como puede observarse, ambas plataformas tienen unas condiciones similares. Además, ambas revisan el contenido manualmente antes de permitir su publicación. La diferencia principal está en la distribución de ganancias entre el vendedor y la página web, donde en Unity Asset Store es de 30% - 70%, y en Fab, de 12% - 88%; respectivamente.

3.3 Recursos requeridos

RECURSOS NECESARIOS PARA REPLICAR EL PROYECTO

- CPU de gama media.
- GPU de gama alta o de gama media si incluye tecnología DLSS.
- Memoria RAM con al menos 8gb.
- Un ratón de ordenador cómodo y con botones especiales para asociar atajos y ganar velocidad en procesos.
- Conexión a Internet.
- Acceso a documentación amplia y referencias.
- Editor de texto y de hojas de cálculo.
- Suscripción premium a Chat GPT (obligatorio a fecha de 2025 por falta de modelos en español).
- Blender.
- Meshy AI 3.
- Editor de imágenes por capas.
- Autodesk 3ds Max.

Estos requisitos tienen en cuenta que se trabaja con muchos modelos renderizándose a la vez por escena para mayor comodidad y velocidad, si se trabaja por asset aislado, los requisitos de hardware se reducen en cierta medida.

RECURSOS OPCIONALES O CONCRETOS QUE SE HAN UTILIZADO

- Procesador Intel Core i5-13600 KF (20 CPUs), 3.5GHz.
- NVIDIA GeForce RTX 4070 Ti, 12gb VRAM.
- 32gb RAM.
- 1000mbps (velocidad de descarga).
- Cuenta de Google (Docs, Sheets, Drive).
- Adobe Photoshop (puede sustituirse por Krita u otro editor de imágenes por capas).
- Adobe Substance Painter (opcional, puede sustituirse por Blender).
- Materialize (puede sustituirse en parte por Photoshop).
- Huion Canvas Pro 13 (puede sustituirse por un ratón).

3.4 Viabilidad e implementación

La viabilidad económica es evidente en este caso, puesto que es el objetivo final del proyecto: vender un asset que pueda posicionarse bien en portales de venta online.

Aún así, el desarrollo del proyecto ha sido mucho más lento de lo que sería rentable por tratarse de la primera vez que el autor se embarca en un trabajo de estas características. Carecía de la mayoría de conocimientos técnicos para llevarlo a cabo, lo que en retrospectiva resultó en cerca de 7 decenas de horas malgastadas en flujos de trabajo erróneos o en generar productos por debajo de los estándares de calidad requeridos. Los estragos del método prueba y error han aparecido fuertemente reflejados en este proyecto.

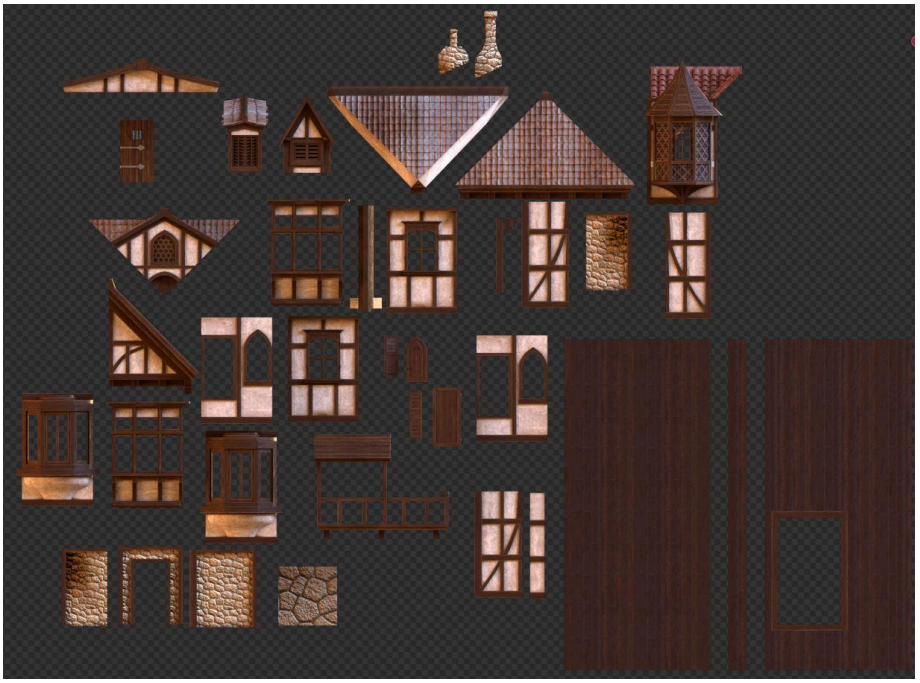
Como punto muy positivo, se estima que una segunda iteración del trabajo se podría llevar a cabo mucho más rápido y resultar rentable económicamente, con un beneficio por hora invertida mucho mayor. Además, el tiempo trabajado en la industria previo a este desarrollo ha ayudado a apuntar necesidades de los estudios que la mayoría de paquetes de assets no tienen en cuenta, como la inclusión de máscaras parciales y decals. Esto puede ayudar a posicionar el producto. No obstante, es cierto que ya hay librerías gratuitas de decals de alta calidad gracias a Epic Games.

Cabe resaltar como conclusión que la venta autónoma de entornos 3D realistas para empresas de videojuegos no es un ámbito particularmente lucrativo por el número reducido de clientes potenciales, pero se espera ver cierto rédito en los meses venideros. En casos particulares de éxito, este tipo de productos ya han demostrado ser capaces de dar pie a la creación de empresas.

3.5 Resultados del proyecto y análisis

Aquí se presentan no solo los resultados obtenidos, sino también las metodologías y conceptos teóricos que surgieron durante las fases finales del proceso. Es decir, se incluye todo aquello que, una vez finalizado el producto, pudo identificarse como una opción más adecuada, útil o contextual, incluso si no resulta determinante, pero sí relevante para comprender el desarrollo completo. Como el proyecto tiene un matiz autorreflexivo sobre los flujos de trabajo, tiene sentido añadir estas metodologías como resultado, no solo el producto en sí.

Figuras 24 y 25 (Debajo). *Producto final en Blender.*



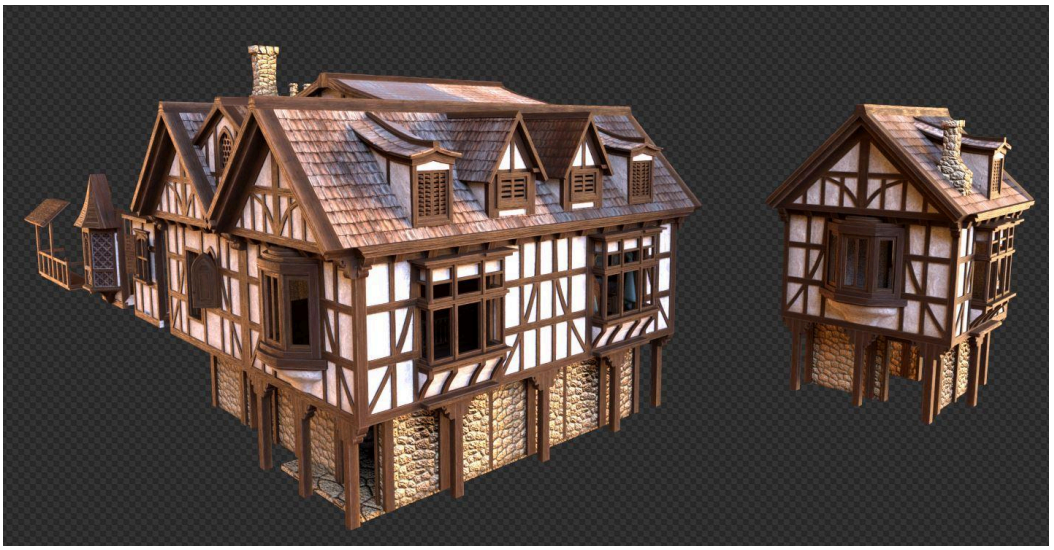


Figura 26. *Ejemplos de casas montadas con los assets modulares arquitectónicos del producto final, Blender.*

Total Props Únicos	Arquitectónicos	Contenedores	De Cocina	Muebles
110	39	7	12	29
Decoración	**Misceláneos**	**Props de Iluminación**	**Variantes por cambio de material**	
7	12	4	9	

Tabla 10. *Modelos resultantes por categoría.*

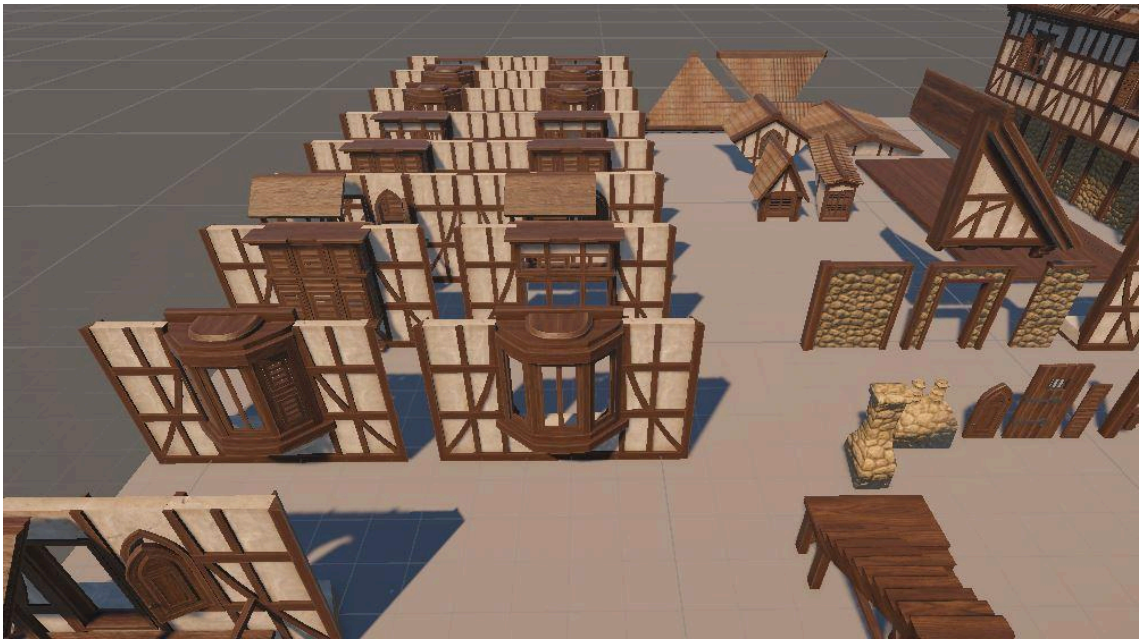
Total Texturas	Texturas Propias	Texturas en base a CC0[30]	Total Materiales	Materiales Maestros	Instancias de Material	Decals	Máscaras
115	91	24	33	3	30	6	3
Texturas con IA	**Texturas Procedurales**	**Texturas Pintadas a Mano**	**Texturas con Materialize[31]**	**Texturas Únicas no Compartidas**			
18	8	12	16	9			

Tabla 11. *Categorización de texturas resultantes del producto.* Enlace con demos en vídeo de los resultados: https://drive.google.com/drive/folders/1g_Tk9fEmIIvUbw4TKRpNAuihHVNpTCye?usp=sharing

[30] **CC0**: Creative Commons Zero. Licencia gratuita sin atribución. Aquí se refiere a material ajeno sin copyright que ha sido modificado para adaptarlo al proyecto.

[31] **Materialize**: programa que calcula mapas de texturizado a raíz del Albedo o Diffuse.

Figuras 27-29. *Renders del producto en Unreal Engine 5.5, con Lumen.*



Figuras 30, 31. *Escena Overview con todos los assets del paquete, Unity 6.*

3.5.1. Estudio de Mercado

Los resultados del escrutinio de los 100 primeros assets de entornos 3D en la lista de popularidad o en la de promocionados en Unity Asset Store y en Fab (los principales portales de venta en la actualidad) son los siguientes:

Por "estilo" en *Fab*

Low Poly	**Estilizado**	**Realista**	**Otros**
5 assets	11 assets	82 assets	2 assets

Tabla 12. *Resultado del estudio de assets por estilo en Fab.*

Por "estilo" en *Unity Asset Store*

Low Poly	**Estilizado**	**Realista**	**Otros**
26 assets	16 assets	58 assets	0 assets

Tabla 13 *Resultado del estudio de assets por estilo en Unity Asset Store.*

Por "ambientación" en *Fab*

Urbano	**Naturaleza**	**Medieval**	**Ciencia-Ficción**	**Guerra**	**Otros**
26 assets	22 assets	21 assets	13 assets	15 assets	3 assets

Tabla 14. *Resultado del estudio de assets por ambientación en Fab.*

Por "ambientación" en *Unity Asset Store*

Urbano	**Naturaleza**	**Medieval**	**Ciencia-Ficción**	**Guerra**	**Otros**
35 assets	21 assets	22 assets	7 assets	12 assets	3 assets

Tabla 15. *Resultado del estudio de assets por ambientación en Unity Asset Store.*

ESTILO

En Fab se encuentran muchísimos más assets realistas que estilizados o low poly, con una mayoría aplastante del 82%. Esto muy probablemente se debe a que era la antigua tienda de Epic Games, los creadores del motor Unreal Engine. De hecho Fab cuenta con integración directa dentro de Unreal. Al tratarse de un motor menos especializado en juegos de teléfono y en cambio, mucho más orientado a juegos grandes, se usa menos en desarrollo indie. Por añadidura, es el predilecto de los grandes estudios de la industria que no cuentan con un motor propio. Además, también es bastante más complejo de usar y requiere de más inversión de recursos en optimización, por lo que para un desarrollo viable en costes suele restringirse Unreal a equipos grandes con mucha formación en su uso y años de experiencia. Estos equipos tienen mucho más presupuesto y, por norma general, pueden permitirse los costes superiores

de los paquetes realistas, tanto por su coste unitario de producto como por la dificultad añadida de adaptarlos a las necesidades del proyecto y el extra requerido en optimización a posteriori (mayor carga poligonal, texturas pesadas, iluminado más complejo, etc). Aún así, es importante tener en cuenta que muchos estudios grandes no compran assets porque los hacen ellos mismos.

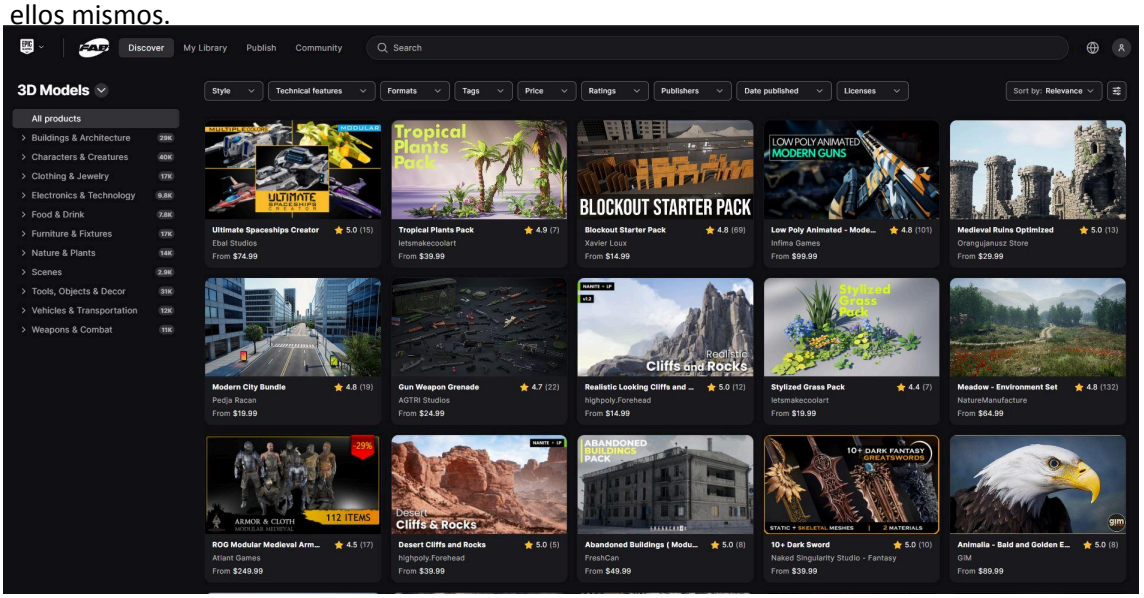


Figura 32. *Página de destacados en Fab.*

En Unity Asset Store, a fecha de marzo de 2025, se observa también una mayoría predominante del estilo realista, aunque no tanto como en Fab. Esto sorprende puesto que se esperaba una dominancia del low poly. Aún así, este se encuentra en segundo lugar. A pesar de que Unity Asset Store coincide a grosso modo con los números de Fab, la diferencia más notable es que en la Asset Store hay más low poly que estilizado. Esto tiene sentido, ya que este portal de venta tiene integración con el motor Unity, que suele ser el motor elegido por equipos de desarrollo 'indie' y para juegos de teléfono móvil. Los juegos indies tienden a contar con menos dinero para la implementación del juego, lo que relega el uso de assets realistas a un segundo lugar, pues tienden a ser más caros. Además, editar assets para que se ajusten a las necesidades del proyecto o para que se acoten en una dirección artística común es mucho más sencillo si se usa low poly, por lo que es accesible a equipos de poca carga artística. Realizar assets realistas suele tener una curva mayor de aprendizaje en comparación. Por otro lado, desarrollar juegos para móvil también complica el uso de assets de estilo realista, pues tienden a ocupar más espacio en disco y a no estar tan bien optimizados para el motor en comparación, ya que en muchas ocasiones, los assets low poly solo usan color plano y un único mapa de texturas por material.

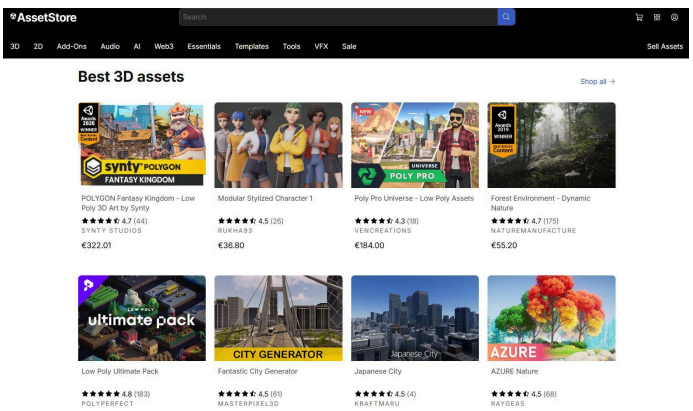


Figura 33. *Apartado "Best Assets" en Unity Asset Store*

Puede observarse que el estilo estilizado no es popular en ninguna de las tiendas. Esto es sencillo de explicar, y es que los gráficos de dibujo animado son altamente dependientes de una línea artística común, cosa que es difícil de respetar si se compran assets sueltos. La mayoría de desarrollos estilizados hacen sus propios assets para asegurar su dirección artística. El realismo es mucho más fácil de combinar siempre que se usen unas resoluciones de textura y texel density similares, y el low poly tiene una línea de estilo fuerte, que aunque pueda notarse más la diferencia entre personajes de distintos paquetes, es mucho menos notorio en entornos y en props.

Se concluye entonces que tanto en la Asset Store como en Fab es altamente más popular publicar assets de estilo 'realista', por una gran diferencia, del 58% y el 82% respectivamente. Podría deducirse también que es más lucrativo, aunque no hay pruebas directas al respecto. Quizá muy pocos paquetes vendan mucho. También hay que tener en cuenta que los costes de estilo 'realista' son algo más altos que los de estilizado y mucho más elevados que los de low poly. Aún así, viendo las incidencias ridículas que tienen el low poly y el estilizado en comparación, parece lógico encauzar el presente proyecto por el estilo realista. Los estudios más grandes puede que compren menos assets, pero si hay tanta presencia de realismo en las tiendas debe de ser porque hay un número potencial suficiente de estudios AA o de equipos indie que también trabajen con estilo realista.

TEMÁTICA

Por temáticas, aunque no por mucho, la más común es la urbana, o sea, ambientación contemporánea y sobria, sin elementos de ficción. La segunda es un empate de media entre ambas plataformas entre naturaleza y medieval. Ciencia Ficción y el resto van bastante a la zaga.

Se concluye que las temáticas más populares y que más arriba posicionan entre los assets más vendidos, son aquellas que tienen un público potencial mayor. También puede darse el caso de que una ambientación poco común venda mucho por no haber casi oferta pero solucionar una demanda real. Aún así, parece más prudente ir sobre seguro en base a números.

Para evitar saturación, se decide no escoger la temática más popular, sino elegir entre las dos que ocupan el segundo puesto. Los assets de naturaleza, aunque vendan mucho, son repetitivos de hacer y usan unas técnicas muy concretas de planos con transparencia que apenas se usan en otros flujos y, por tanto, si el objetivo es aprender al máximo sobre arte 3D mientras se desarrolla el proyecto, no son la mejor opción.

Entonces, descartados ya los assets de naturaleza, la opción más viable es la ambientación "medieval".

CONCLUSIÓN

Se hará, en base al análisis previo, un asset “realista” de entornos 3D “medievales”. Para tocar el mayor número posible de flujos de trabajo, se harán tanto props pequeños, como props grandes y protagonistas, como edificios.

Una forma de aunar todo esto y que a la vez coincide con un setting bastante demandado es un entorno de “taberna de fantasía”.

3.5.2. Modelado

3.5.2.1 Geometría Cosida y Flotante

Se comprobó que efectivamente modelar con geometría flotante es mucho más rápido, pero la parte de que la geometría es menor si no se cose la pieza resultó ser en realidad un tema muy diferente. Colocar la geometría flotante según se necesita en el momento aumenta mucho el número de vértices, lo que no necesariamente se traduce en caras extra, pero sí que aumenta el uso de memoria de vídeo (VRam). Aunque el número de caras sea menor, si por culpa de la geometría flotante el número de vértices es desproporcionado, el rendimiento puede ser peor.

Es necesario planear bien la geometría flotante para que realmente se reduzca la geometría. No coser los vértices, de por sí, no reduce el polycount.

Tras la introducción de modelos con geometría flotante o “loose geometry” en motores gráficos se notó un error de iluminación aparentemente grave en la mayoría de ellos, a pesar de no ser perceptible en Blender o en 3Ds Max. Se trata de que en piezas grandes en unidades de tamaño se generan artefactos de iluminación cuando hay geometría colindante que no está cosida. Esto es especialmente acusado en aristas que coinciden, pero que no comparten la posición de sus vértices. En caso de que los vértices estén duplicados en el lugar exacto, estos problemas de iluminación se notan algo menos. Es recomendable entonces coser las piezas más grandes o intentar que al menos no tengan aristas que ocupen el mismo lugar si no están cosidas. En modelos pequeños apenas se percibe y no hay problema, pero en piezas grandes pueden generarse errores en las sombras proyectadas (con huecos donde las aristas coinciden) aún cuando el modelo en sí se ve bien.

No obstante, estos errores se deben principalmente a la iluminación en tiempo real y se quitan al reducir el “bias” de las normales en las opciones de luz del motor. La mayoría, además desaparecen al bakear las luces, aún con el bias alto. Es un defecto común, pero no es de recibo para un asset profesional porque el cliente podría querer usar iluminación en tiempo real. Los revisores manuales de la Asset Store y de Fab podrían no aprobar el asset.

Figura 34. (Izquierda) *Artefactos en sombras aunque el modelo en sí se visualiza bien.*

Figura 35. (Derecha) *Artefactos entre aristas que coinciden pero no están cosidas.*

3.5.2.2 Elaboración de High Polys

Se probó inicialmente a generar high polys de 11 assets, 1 de ellos un prop y el resto partes de edificio, y se concluyó que es preferible no hacer versiones subdivididas para un paquete medieval, solo aumenta exponencialmente el tiempo de desarrollo sin beneficios aparentes en muchos de los casos. La grandísima mayoría del detalle se puede obtener con texturas genéricas en modelos tan orgánicos. Además, el bakeado de Ambient Occlusion es más limpio si se selecciona la opción "Use low poly asset as high poly version" que ofrece Substance Painter. El bakeado de geometría real a veces genera artefactos en los ejes duros, cosa que la opción comentada nunca crea, y cuando ambos métodos funcionan adecuadamente, apenas sí hay diferencia perceptible. Es cierto que los artefactos se pueden evitar haciendo pruebas con diferentes padding o cambiando los seams de lugar, pero si hacer bakeado de high a low lleva varias horas por cada modelo medianamente complejo, mientras que el otro método es automático y no se aprecia casi ninguna diferencia, es un método superfluo.

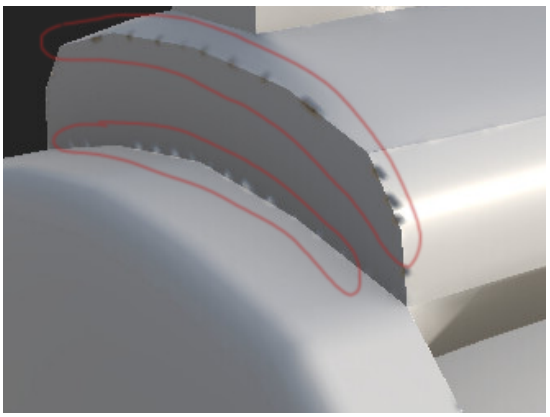


Figura 36. *Artefactos al bakear highs en Substance Painter*

Hay excepciones en las que algunos props requieren meticulosos detalles que son difíciles de sacar con texturas genéricas, ya sea por la necesidad de una textura muy concreta que no se puede conseguir, o simplemente porque se pretenda hacer algo único. Para estos casos, si el número de materiales únicos no es una preocupación grande, sí que se puede hacer una versión high poly del modelo con estos nuevos detalles para transferirlos. Eso sí, esto impediría el texturizado por "overlap" y requeriría de un material único por asset (más draw calls en el motor), por lo que es preferible no hacer versiones high poly del asset en sí, sino hacerse una "trim sheet" reutilizable gracias a detalles en high poly que se le apliquen a un "plano" (*ver más adelante*).

Esta trim sheet también tiene su parte mala, y es que al ser común entre varios assets, impide bakear "ambient occlusion" (AO) real a cada modelo por separado. Entonces, elaborar high polys para assets únicos solo es recomendable si el modelo va a ser visto de cerca por el jugador o es muy grande, y por tanto, texturizar con overlap para mayor optimización no es una opción porque usar ambient occlusion genérico lo hace verse cutre. Solo en este último caso en el que no se puede hacer AO genérico y el asset se va a ver de cerca es viable hacer

high poly de un modelo entero. Los assets directamente arquitectónicos como paredes o tejados no cumplen esta regla, son una excepción. Como son objetos estáticos que no van a ser movidos en el videojuego se les puede hacer otros trucos después en el motor (*ver más adelante*).

La siguiente tabla resume lo descrito de forma clara:

¿El modelo se ve de cerca y/o es grande y no es un edificio?	**¿Se puede permitir crear un material y texturas únicas?**	**¿Requiere detalle único no obtenible con texturas genéricas?**	**¿Es entonces recomendable hacerle un High Poly?**	**Tipo de Bake**
Sí	No	No	No	Ningún bake, texturizar por overlap reutilizando materiales.
Sí	No	Sí	No	Modelar detalles high poly sobre un plano para luego bakearlo y tener una trim sheet reutilizable.
Sí	Sí	Sí	No	Hacer un bake de la versión high poly a la low poly. Texturizar como asset único.
No	No importa, ya no se va a hacer de cerca. Saltar al tipo de bake.	–	–	Ningún bake, texturizar por overlap reutilizando materiales.

Tabla 16. *Cuándo hacer una versión high poly de un prop y cuándo no.*

3.5.2.3 Render vs Juegos

Una cosa que muchos assets tienen en común cuando vienen de un modelador genérico es un uso excesivo de geometría. Haciendo ingeniería inversa a paquetes de la tienda se ha visto que incluso aquellos que se anuncian como para videojuegos pueden venir con loops de soporte adicionales para subdivisión. Esto no es necesario, los assets de equipos muy reconocidos solo tienen geometría que contribuya a la geometría, todo el resto lo eliminan. Incluso se ha comprobado como cierran tapas de cilindros con un simple “collapse to center” en lugar de mantener una topología limpia como sería un “grid fill” con un refuerzo previo del eje.

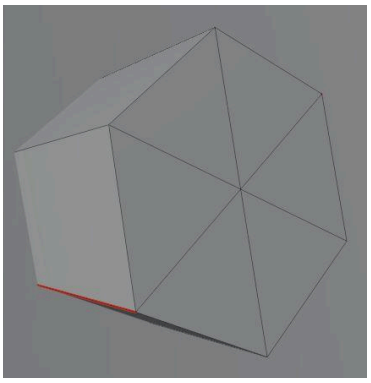


Figura 37. *Tapa colapsada.*

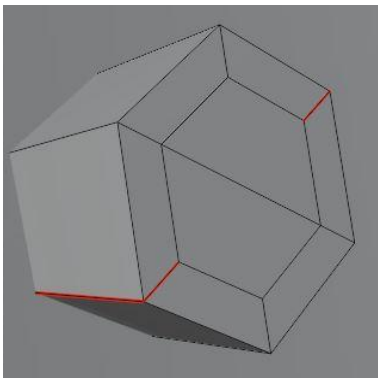


Figura 38. *Tapa sin colapsar.*

De cara a este proyecto, se ha decidido reducir el conteo de polígonos al mínimo aunque la topología resultante sea horrible; siempre que no se trate de partes protagonistas de un modelo. En este último caso se ha mantenido topología limpia para evitar posibles artefactos en bakeados en el motor, aunque son extremadamente raros en este caso. Solo en 1 de los 10 bakeados de prueba hechos en Unity built-in render pipeline se ha producido un error claramente visible debido a mala topología. Debajo se muestra un ejemplo de este artefacto que ya fue detectado incluso desde el programa de 3D, aún cuando todas las caras de la tulipa de la lámpara estaban suavizadas en el mismo grupo (jerga de Maya y 3dsMax) o bajo el mismo ángulo de suavizado (en jerga de Blender), se generaban artefactos. Por supuesto, no había caras duplicadas ni volteadas, pero el suavizado dejaba en evidencia el colapso de vértices sin uniformidad de malla (ver Fig.).

Figura 39. *Artefactos al colapsar vértices al centro.*

Como puede verse, es un problema sutil que no se aprecia desde lejos, especialmente después del texturizado. Entonces, en este prop en concreto, que no es muy grande y no se prevé que aparezca cerca en cámara, se dejó así para reducir el polycount. En otros casos, si se tratara de una zona grande en un asset protagonista, sería conveniente reducir la geometría sin recurrir a colapso, manteniendo la silueta con un soporte y usando un relleno de quads en matriz.

Otro motivo para añadir geometría extra donde no se necesita es evitar "shading weirdness", un error de iluminación que puede ocurrir en zonas coplanarias limpias, pero que tienen poca geometría y son muy grandes en unidades de tamaño. En este proyecto no ha ocurrido, pero se ha añadido geometría extra para mantener cierta uniformidad en la densidad de la malla por precaución en los props arquitectónicos, que son los más grandes. Se ha intentado que afectara lo mínimo posible al polycount, pero se ha hecho para evitar futuros errores en baking en los motores.

Por último, existe un motivo de peso para mantener uniforme la densidad de una malla aunque se añada geometría aparentemente inútil: posterior aplicación de "vertex paint" en los motores. Esta técnica se ha dejado fuera del alcance de este proyecto, pero se trata de un sistema muy usado en los juegos AAA. Permite almacenar información en los vértices del modelo. Normalmente se emplea para mezclar varios materiales en un modelo y diferenciar así assets que anteriormente eran iguales. Vertex paint, para funcionar bien requiere de cierta subdivisión de la malla, es decir, si no tiene los suficientes polígonos en una zona porque hay caras muy grandes, no se puede aplicar.

3.5.2.4 Geometría por prop

Una vez más, tras haber comprobado muchos ejemplos, se ha llegado a otra conclusión: por mucho que se trate de un juego AAA y que el prop deba tener calidad y se vaya a ver de cerca, si se va a repetir muchas veces en escena ha de tener geometría mínima. El ejemplo es el barril. Se trata de un prop que se suele iterar mucho para decorar y que es fácilmente reconocible con poca geometría, por lo que en los paquetes de assets siempre tiene poca geometría.

El barril modelado para el proyecto contaba con 2700 triángulos, y eso que ya se había modelado de forma conservadora, pero aún así, en los ejemplos reales se veía como nunca superaba los 1000 y tendía a oscilar entre los 500 y los 700 tris.

Para salir de dudas y hacer un paquete competitivo, se escanearon por IA generativa de texto y análisis los datos públicos acerca de assets de los juegos The Witcher III (algo anticuado en otros aspectos, pero aún muy vanguardista de cara a producir assets 3D) y Hogwarts Legacy (ejemplo moderno de excelencia) para determinar medias recomendadas de polígonos por tipo de asset para un juego AAA. Los resultados son los siguientes.

Categoría funcional	Descripción	Recomendación de triángulos
Mobiliario utilitario	Mesas, sillas, bancos, camas, taburetes, estanterías vacías	800 – 3.000
Mobiliario decorativo	Estanterías llenas, muebles ornamentales, repisas, carros de mano, soportes	1.500 – 4.000
Almacenamiento simple	Cajas, barriles, sacos, cofres cerrados sin apertura	300 – 1.200
Almacenamiento interactivo	Cofres con apertura, compartimentos, baúles complejos	2.000 – 4.500
Contenedores menores	Botellas, tarros, cántaros, cuencos, copas, jarras	200 – 800
Utensilios y menaje	Platos, cubiertos, herramientas pequeñas, morteros	200 – 600
Iluminación y fuego	Antorchas, candelabros, faroles, braseros	800 – 2.000
Decoración temática	Trofeos, banderas, tapices, escudos decorativos, estatuillas, tótems	1.000 – 3.000
Props destructibles	Barriles rompibles, cajas con físicas, sacos que se deforman	1.500 – 4.000
Mecanismos de interacción	Palancas, interruptores, ruedas giratorias, tapas móviles	2.000 – 5.000

Tabla 17. *Recomendación de polígonos por asset en base a The Witcher III y Hogwarts Legacy.*

Con esto en cuenta se han revisado todos los props del proyecto, habiéndose requerido cambios estructurales grandes e incluso un nuevo texturizado en los siguientes

props: *reinforcedChest, barrel, woodenBucket, roundedTable, vanityTable, reinforcedHangingShelves, fancyChair, hangingLamp, slattedWoodenCrate.*

3.5.2.5 IA en Modelado

Aunque existen otras IAs de texturizado, solo se ha analizado Meshy AI 3 por ser la más avanzada hasta la fecha; y con diferencia.

Modela muy rápido y con mucha fidelidad a las imágenes de input, pero a día de hoy aún genera incongruencias entre las diferentes vistas del modelo, generando modelos que no cosen bien en los costados.

Para ser utilizada correctamente requiere de imágenes sombreadas, ya que parece ser cómo interpreta los volúmenes: en base a las diferencias en escala de valor. Funciona con concepts dibujados a mano, pero solo si el sombreado está lo suficientemente elaborado, no funciona solo con siluetas o line-art. La iluminación del dibujo ha de estar lo suficientemente trabajada. Se pueden utilizar generaciones de imagen de otras IAs como referencia para Meshy y el resultado es bueno.

La fidelidad a la referencia es muy buena. A diferencia de otras IAs que escanean la imagen para obtener un prompt a raíz del cuál generar, Meshy hace una rotoscopia real midiendo las distancias entre los puntos clave de la referencia, consiguiendo un alto parecido.

Aún así, todavía no interpreta bien qué partes son sueltas y cuáles están adheridas, es decir, puede no diferenciar entre una tela suelta y una pierna. Cuantos menos adornos colgantes que se interpongan en las articulaciones se le den de referencia, mejor.

Los modelos generados son bastante decentes vistos de lejos, pero requieren de una modificación manual exhaustiva para aguantar visionados de cerca si la intención es obtener un modelo de estilo realista. Funciona mejor para low poly. Además de los ya comentados errores entre vistas, la topología generada es sucia. Se ha comprobado que de todos modos es sustancialmente mejor que la del año pasado (2024), pero sigue necesitando una limpieza notable por parte del artista. En el caso que a este proyecto atañe (hard surface), donde la topología es mucho menos importante, los resultados son bastante mejores.

No obstante, aunque las generaciones no estén a la altura de un producto final, la IA en modelado puede utilizarse como malla base para acelerar el flujo de trabajo. De forma casi instantánea se tiene un modelo muy similar al concept, que con un par de recalculaciones de malla o “remeshes”, se puede emplear como un buen punto de partida.

A nivel personal, no se recomienda Meshy AI por su modelo de negocio, que a fecha de hoy parece algo predatorio. En lugar de contar con una suscripción mensual como otros modelos de IA, funciona por créditos. Esto quiere decir que cada generación cuesta dinero, y como normalmente se requieren varias por producto para ir afinando los resultados, termina por ser económicamente poco rentable si en el equipo de desarrollo hay artistas 3D. En cambio, para casos en los que no haya ningún perfil artístico en el equipo, sí que puede ser una buena solución para elaborar productos 3D. Como recomendación final, se sugiere esperar a un cambio de modelo de negocio a tipo suscripción.

	Meshy AI
Rapidez	Instantánea.
Fidelidad Visual	Espléndida, rotoscopia la referencia para crear el parecido.
Congruencia	Genera diferencias incoherentes entre vistas.
Topología	Aún algo pobre y requiere de corrección manual, pero cada vez mejor.
Nº iteraciones	En la mayoría de casos es preferible hacer varias generaciones (2-5).
Economía	Modelo de pago por créditos, poco rentable a fecha de 2025.

Tabla 18. *Meshy AI en modelado.*

No se adjuntan imágenes de resultados porque no se ha llegado a utilizar Meshy para los productos finales en este proyecto. Se ha analizado su viabilidad mediante varios intentos, pero se ha descartado finalmente tras considerar que hacerlo a mano desde cero era más rápido que corregir errores. Esto es así porque se trata de modelos hard surface con optimización en número de polígonos para videojuegos. En un caso de uso donde se requiera crear personajes, sí que se estima óptimo partir de mallas base hechas en Meshy.

Otra ayuda complementaria para automatizar procesos de modelado es la simulación de físicas para textiles. Como es algo más nicho, se expone en el Anexo.

3.5.3 Texturizado

Mediante la observación de casos reales de productos finales similares a lo que se busca obtener, y gracias a cierta ingeniería inversa generalizada, se ha llegado a la siguiente conclusión: el flujo estándar de texturizado no es que esté obsoleto como se esperaba, sino que se combinan muchos flujos diferentes y con calidades resultantes totalmentes distintas en función de qué prop es el que se va a texturizar. Es decir, no existe un flujo moderno diferente, más bien se trata de una combinación de antiguos y nuevos por motivos de optimización.

La idea general es que siempre se intentan condensar mapas en el menor número posible de espacio y representar el menor detalle cuando el asset no es protagonista, y por el contrario, aumentar la calidad para assets que van a estar a mayor tamaño y que van a aparecer más en pantalla. En función de estos dos approaches y en qué cantidad se considera que el prop encaja con uno de ellos, se utilizan hasta 4 flujos diferentes de trabajo. Estos, de menor a mayor calidad resultante son: la condensación en atlas, el tileado, el uso de trimsheets y el texturizado por material único con bakeados.

Todos los flujos pueden combinarse en mayor o menor medida entre ellos, y a efectos teóricos todos pueden conseguir la misma calidad de producto. Se ordenan así por su uso

práctico optimizado, y es que, por ejemplo, texturizar a gran calidad en atlas requiere de un tamaño de imagen absurdamente grande, o texturizar de manera única props muy pequeños sería un gasto desmedido de draw calls y tamaño en disco de texturas cuando no sería necesario.

También, para cada uno de estos flujos de trabajo, la forma de texturizar puede ser diferente, y es que las texturas tileables suelen hacerse de manera procedural, los atlas tienden a ser texturas basadas en imágenes, etc. Por supuesto, para casos concretos o por motivos de diseño, se pueden combinar o alternar todos los flujos para cualquiera de los approaches.

3.5.3.1 **Texturizar assets arquitectónicos:** modularidad

TEORÍA

A la hora de texturizar un edificio, una vez considerado su gran tamaño y protagonismo en cámara, cae de cajón que el detalle debe ser máximo. ¿Cuál es el flujo de trabajo que obtiene mayor calidad? Resulta que es el texturizado por material único con bakeado de Ambient Occlusion.

Este flujo funcionaría sin problema en otros ámbitos, pero en videojuegos no, y es que conseguir tanto detalle a un texel density muy grande (un edificio es grande en tamaño) es inviable a nivel técnico. Esto requeriría de unas texturas gigantescas de 8K, 16K o incluso más. Los motores gráficos de videojuegos no admiten en casos normales texturas superiores a 8K, por no hablar del tamaño en disco que tiene eso. Es bastante más efectivo para ahorrar en peso de archivos el hecho de reutilizar materiales entre props.

A nivel de optimización de tiempo de trabajo, texturizar así un edificio también es inabarcable, y es que la mayoría de materiales que va a llevar esa textura gigante única se podrían reutilizar de otros props, pero al no poder haber overlap en las UVs para calcular Ambient Occlusion y el resto de mapas de detalle que se proyectan sobre la textura, no es posible reutilizarlos; es necesario hacerlos de nuevo.

Otra consideración es la reutilización del asset en sí. El cliente probablemente prefiera comprar un paquete a raíz del cuál pueda crear edificios diferentes, en lugar de tener un único edificio disponible. Da poco juego y haría falta comprar muchos paquetes para construir una escena entera, paquetes que muy probablemente tampoco coincidan en línea artística. Es mucho mejor ofrecer un edificio con cierta modularidad para que el cliente monte el edificio que quiera y así lo pueda reutilizar para más casos. Esto impide el flujo de texturizado por material único con ambient occlusion, y es que al no conocer la disposición final de las piezas modulares que el cliente va a tener, bakear ambient occlusion al edificio haría no coincidir el sombreado entre piezas.

Con esto en mente, podría parecer que la mejor solución es el tilear texturas preparadas para ello, pues al poder repetirse varias veces a lo largo del modelo se consigue un texel density lo alto que se desee. Esto, en realidad, conlleva dos problemas cuando se texturiza un asset muy grande en unidades de medida. El primero es que haría falta tilear muchas veces la misma textura para llegar al texel density acordado, lo que haría que se

terminara viendo dónde se repite la textura. Preparando bien la textura es posible minimizar este efecto, pero siempre termina por aparecer si la superficie a cubrir es lo suficientemente grande. El otro inconveniente es que con texturas tileables no se alcanza el detalle necesario como para algo tan grande, se vería “repetitivo”, sin variaciones.

La solución es ofrecer un edificio modular texturizado con trim sheets. Esto, como ya se explica en el apartado 2. Antecedentes, es una textura grande que contiene muchas otras, como un atlas, pero el objetivo es ofrecer por zonas detalles únicos dentro de un mismo material para texturizar un único prop o props similares entre sí, no condensar muchos props diferentes en una única textura. Normalmente, se pinta un mismo material en varias zonas, pero con variaciones, para poder usar cada zona en partes diferentes del prop y que no se repita visualmente. También es común disponer las franjas de manera solo horizontal o solo vertical para que puedan ser tileables al menos en uno de los ejes, puesto que la naturaleza de una trim sheet impide el tileado completo. Esta tileabilidad por eje singular es suficiente para iterar detalles como tablones de los que no se conoce la longitud, cenefas ornamentales, grabados, etc. Esto combina lo mejor de ambos mundos, la posibilidad del tileado y la posibilidad de la variación.

Una trim sheet tiene la ventaja de ofrecer el suficiente detalle percibido como único sin requerir de una textura gigante, aunque sí que debería ser una textura grande, por ejemplo 4K.

RESULTADOS

Para diseñar una trim sheet de la taberna era necesario incluir tablones de madera y el yeso de la pared. Como se trata de la parte más visible del paquete de assets, y por ende, su protagonista o hero prop, se intentó alejarse de materiales preconcebidos o en base a imágenes. Para ello, se optó por el texturizado procedural, que siempre ofrece mayor diferenciación cuando se trabaja con muchas capas.

Con el objetivo de texturizar una trim sheet en Substance Painter, se exporta desde Blender un plano sobre el que trabajar. Una vez en Substance, al no haber geometría, era imposible bakear mapas, y aunque el AO tampoco se pretendía usar porque estropea la reutilización, sí que se querían utilizar aquellos mapas que son útiles para hacer máscaras. Si no se pueden generar desgastes y demás, no tiene sentido recurrir a una trim sheet, se puede optar directamente por texturas tileables por material único. Entonces, se tuvo la idea de exportar el plano, pero con ciertas extrusiones y hendiduras allí donde se planificaba colocar los tablones de madera. A prueba y error, se dispusieron tablas de cuatro grosores diferentes y con al menos dos variantes por tipo; los delgados con hasta cinco. Al haber volumen en el plano, ahora se pudieron generar desperfectos en los bordes y marcar las profundidades.

También se aprovechó para falsear el AO en el material de yeso. Para ello se dispusieron tres franjas con planchas de yeso a diferentes tamaños y con variaciones. A cada una de estas planchas se les añadió una capa de pintura en marrón oscuro saturado usando máscaras en base a curvature. Estas sombras cubrían los bordes donde los tablones ocultarían la luz. Así, se imitaba el cálculo de la iluminación sobre la textura como haría AO, pero de manera reutilizable, para posteriormente en Blender colocar las caras a mano sobre estas

planchas de material de yeso, haciéndolas coincidir a placer. Por último, se aplicó un filtro "enfocar más" en Photoshop al Albedo y al mapa de normales para crear falsa sensación de estar representados a una resolución mayor.

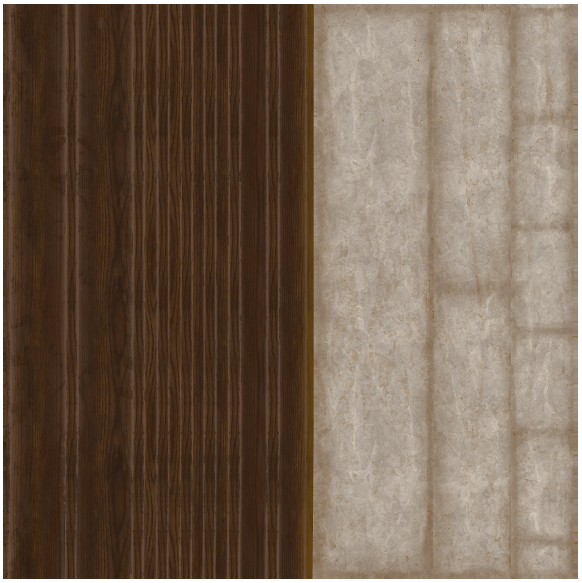


Figura 40. *Albedo del trim sheet usado para texturizar la taberna.*

A pesar de no haber AO real en los mapas de texturizado, este se puede calcular más tarde en el motor, por lo que el cliente no espera que venga hecho si los assets son modulares. Una vez montado el edificio a placer, el comprador puede calcular AO en el motor gráfico , ya sea en tiempo real gracias a tecnologías como SSAO, o porque se haya bakeado la iluminación en la escena de forma estática, lo que genera un AO percibido gracias a los lightmaps.

3.5.3.2 **Texturizar props únicos:** proceduralidad (FORMATO)

Los props de carácter único que o bien se quieren diferenciar del resto o no se pueden texturizar por imágenes basadas en fotografías o pintadas a mano (mipmap textures), se hacen proceduralmente. A veces es porque la geometría resultante es difícil de adaptar a una textura prefabricada, y otras porque directamente no existe o no se encuentra una textura adecuada.

Teóricamente se crean proceduralmente para máximo detalle, pero en base a la experiencia recabada en el proyecto, el nivel alto de calidad se puede alcanzar de muchas otras maneras. De hecho, las texturas que vienen de la fotogrametría con licencia creative commons zero, muy usadas por freelancers, son de una calidad excelente y a veces hasta mejor que aquello que se pueda obtener por nodos.

Al final, los props grandes que se planearon para proceduralidad en Substance Painter, se pudieron hacer por mipmap textures con un nivel de detalle deseado, pero sí que hubo casos inesperados que requirieron de proceduralidad porque tenían geometrías muy extrañas.

Geometrías que se beneficiaban mejor de un pintado por capas y un control manual en Substance Painter. Estos fueron los casos concretos del cuerno de beber y la calavera de cabra.

Para el cráneo de cabra, el ejemplo que se va a detallar aquí, se creó una versión high poly con la intención de transferir detalle, a la que no hizo falta sacarle UVs (solo se usó para proyectar geometría). Luego, colocando la versión low y la high en el mismo lugar exacto y teniendo cuidado de que no hubiera una diferencia grande de tamaños entre la versión subdividida y la original (se pierde tamaño al subdividir porque se redondea), se exportaron ambos desde Blender en FBX. Después, se importó como proyecto 1K, pues no era necesario más para ese tamaño. Se procedió a bakear AO y los mapas necesarios para calcular máscaras inteligentes, pero se encontraron dos errores importantes.

Figura 41. *Versión high poly vs low poly del cráneo de cabra.*

El primer error fue haber exportado el modelo con aristas duras, es decir, marcadas como “sharp”, que resulta que tras una investigación exhaustiva se ha encontrado que en Substance casi siempre dan artefactos cuando no coinciden con los seams de las UVs. Entonces, una arista sharp ha de coincidir con un split UV (un seam). Esto, Substance tiende a detectarlo en la previsualización del bake y marca las aristas problemáticas en morado (ver Fig.). Este conflicto también se aplica a la generación de los lightmaps en motores, pero en mucha menor medida, la probabilidad de error por arista dura sin split UV en UNity y Unreal es baja. En tan solo una de las diez pruebas realizadas se encontró un artefacto debido a esto.

El segundo error es específico de Blender por cómo calcula las normales; ver sección 3.2.13. Al usar auto-smooth, que calcula el suavizado por cambio de ángulo entre caras, en lugar de por grupos como hacen otros programas, Substance no lo reconoce y puede dar artefactos. La solución si no hay una versión high poly de la que transferir detalles es usar un suavizado “smooth” convencional, que es transversal a todo software, y marcar las aristas duras con “mark sharp” haciendo que coincidan con los seams para no repetir el primer error. Si se dispone de software de Autodesk, no hace falta marcar nada como sharp, sirve con aplicar las caras colindantes a aristas que deben ser duras a grupos de suavizado diferente; que es menos engorroso que hacerlas coincidir con los seams. Luego, si se tiene una versión high poly o una “cage” (ver más adelante), no es necesario nada de esto, vale con tan solo exportar el modelo entero suavizado como “smooth”, y ya es el bakeado desde la versión high la que transfiere la información del smoothing.

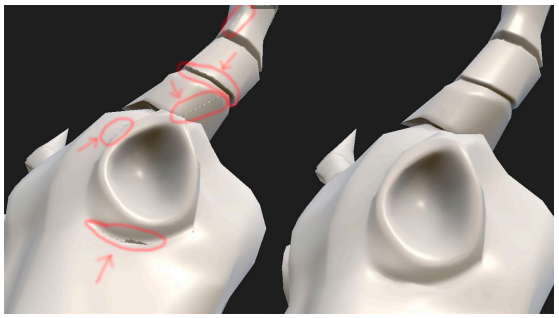


Figura 42. *Artefactos en el bakeado en Substance por usar aristas duras y auto-smooth (izquierda) vs su versión arreglada sin aristas duras y en smooth convencional (derecha).*

Para obtener el bakeado (ver metodología en el apartado 3.2.7) es importante explicar los siguientes conceptos, sobre los que también se desglosan las conclusiones obtenidas tras prueba y error en muchos casos:

En la pestaña Mesh Map Settings:

- **Output Size:** ha de coincidir con la resolución del proyecto. Por defecto no suele coincidir, hay que introducirlo manualmente.
- **Dilation Width:** por defecto en 32px, es la separación para evitar sangrados entre las islas de UVs. Si se han hecho bien las UVs, para resolución 1K se ha de poner entre 8 y 16px, para 2K entre 16 y 32px, y para 4K hasta 64px. El valor más seguro en la mayoría de casos es entre 16 y 32px.
- **Use Low Poly Mesh as High Poly Mesh:** muy recomendado para modelos inorgánicos complejos, con los que a veces da mejor resultado incluso que el bakeo de high polys. Aguanta mejor las aristas duras. Si aún se generan artefactos, se puede usar cage (ver debajo).
- **Antialiasing**: genera artefactos cuando se bakea un modelo que usa ID Map para separar los materiales (flujo que permite aplicar materiales diferentes a un mismo modelo, pero manteniéndolos en una misma textura en lugar de en una por material). Muy recomendado si no se usa ID Map y se busca realismo, ya que genera mejores transiciones en los mapas, pero suaviza los detalles pequeños. Un buen equilibrio es usar 4x o 16x.
- **Cage**: el cage es opcional, pero solo recomendado si se requiere precisión máxima o corregir errores no subsanables de otra forma, pues requiere de generar y exportar más archivos. Se trata de una malla intermedia de tamaño entre el high y el low. Se hace a raíz de duplicar el low y escalarlo (aplicando escala después) de forma que cubra el high poly por completo, pero esté dentro del low. Interpola los rayos redirigiéndolos para mayor precisión. Esto puede arreglar sangrados en aristas suavizadas con el auto-smooth de Blender y solucionar mallas que dan artefactos por ser muy complejas, pero añade tiempo considerable al proceso.
- **Max Frontal Distance** y **Max Rear Distance**: permiten regular la separación entre las mallas para calcular el bake. Una vez cargado el high poly se ven en la ventana principal las mallas con transparencia. Normalmente interesa modular solo la Max Frontal

Distance. El punto dulce de distancia es el justo para que todo salga en azul y beige, pero no salga nada en rojo (ver Fig.).

En la pestaña Ambient Occlusion, Curvature y Thickness:

- **Secondary Rays:** cuanto mayor el número, mejor suavizado y resultado general de los mapas si sale bien el bake, pero mayor probabilidad de errores. Es mejor dejarlo bajo si la geometría es compleja o si las UVs están muy juntas

Para obtener estos resultados se han hecho numerosas pruebas. Entre ellas la propia taberna, que inicialmente se planteó como pieza única a la que bakear AO. Para ello se ha probado con versiones high poly, usando el low como high y también con cages.

Para el cráneo de cabra se utilizó output size de 1024x1024 por ser la resolución del proyecto, una dilation width de 15px por ser 1K y porque las islas de UVs estaban algo separadas, secondary rays a 64, la versión high poly del modelo en High Definition Meshes, desmarcada la opción de Use Low Poly Mesh as High Poly Mesh, ninguna cage cargada, una Max Frontal Distance de 0.011, la Max Rear Distance por defecto, y Supersampling 4x en Antialiasing.

Figura 43. *Max Front Distance insuficiente (izquierda) vs óptima (derecha).*

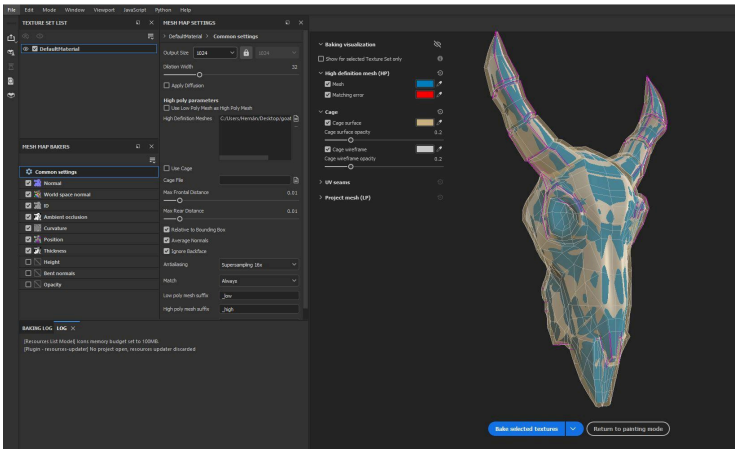


Figura 44. *Interfaz de Bakeo en Substance Painter donde se ve cómo se marcan en morado las aristas que van a dar problemas en el bake. En este caso es porque están marcadas como sharp.*

Figura 45. *Bakeo final del prop "animalSkull". El espacio vacío es intencional para mantener coherente el texel density.*

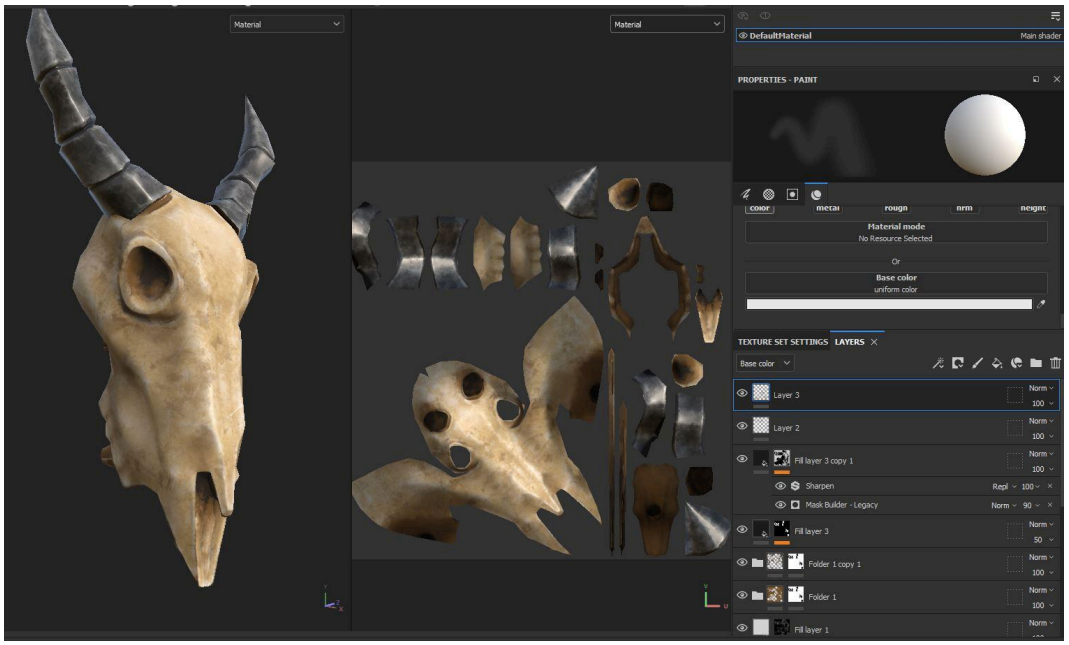


Figura 46. *Resultado final del cráneo en Substance Painter.*

El resultado final se ha obtenido con el material Bone Stylized como base. Luego, con máscaras inteligentes para cavidades y salientes a diferentes tamaños y colores se ha envejecido el cráneo y resaltado detalles. En concreto se han encadenado varias Soft Damage y Dirt Cavities a diferentes tamaños y colores. Para los cuernos, lo mismo, pero en negro, con menos grunge y con el roughness más bajo. Los detalles finales con capas de pintura a pincel manual en marrón oscuro para las sombras y blanco deslucido para las luces. Así se obtuvieron los dientes y otros detalles extras para conseguir mayor credibilidad de volúmenes.

3.5.3.3 **Texturizar props medianos:** texturas tileables

ERROR INICIAL Y RAZONAMIENTO

En un principio, la mayoría de los props ya habían sido texturizados por el método estándar para mayor calidad, que es el texturizado individual, con cada modelo teniendo sus propias texturas específicas y a gran tamaño.

Tiempo después, al hacer ingeniería inversa a algunos paquetes de assets en venta en la tienda, y tras leer varios manuales de uso, se percibió que para una mayor optimización es preferible usar texturas compartidas entre varios props; aún cuando se trata de modelos de calidad AAA. El tamaño en disco de tanto prop por cada paquete importado a un proyecto de videojuegos sería ridículo de lo contrario.

Entonces, se hizo una categorización de los props en desarrollo por su tamaño en cámara. Esto dió como resultado a la mayoría de ellos, es decir, a todos menos al carromato, el tonel grande y, por supuesto, los assets arquitectónicos que componen la taberna.

Al no tratarse de assets arquitectónicos, no había grandes superficies lisas que clamaran a los cuatro vientos que se estaba tileando una textura: hay bastante margen para ocultar dónde se repite. Los props tienden a tener más salientes por unidad de medida y es fácil usarlos para ocultar el tileado. Además, el hecho de compartir materiales ayudó mucho a dar cohesión visual al paquete, los assets parecían pertenecer a una misma temática.

El hecho de tilear texturas fuera del espacio UV 0-1 también facilita usar imágenes de menor resolución, dado que se puede repetir una misma textura varias veces.

PROCESO FINAL EMPLEADO

El flujo, tras prueba y error, ha sido el siguiente:

1. Comprobar que al hacer unwrap del modelo este se reparte de la forma equitativa esperada (con tamaños acordes entre piezas y sin distorsiones imprevistas). De no ser así, buscar qué arista ha sido pasada por alto al marcar los seams. Marcarla como seam y repetir el despliegue de UVs.
2. Mediante el addon de Blender “TexTools”, aplicar a todas las islas de UVs el texel density recomendado para la resolución que se esté empleando (ver Tabla 13). Si la textura tiene unas métricas diferentes o se ha de cuadrar con otro prop, ajustar el tileado a simple vista, con cuidado.
3. Rotar y escalar las islas para poder colocarlas sobre la textura siguiendo los ángulos y distribución que se pretenda conseguir para texturizar correctamente el prop en cuestión. Se ha de tener especial cuidado con el escalado de las islas, ya que se permite y hasta se espera cierta variación en el texel density cuando se comparten texturas (sería imposible de lo contrario), pero no han de exceder una variación de un máximo

del 15% en tamaño. Algunas empresas no permiten más del 5% en sus hero props o en producciones AAA (Polycount, 2023).

4. Comprobar que hay cierto margen entre islas, dentro de lo posible, y con respeto a los bordes del espacio UV 0-1 si es que no se sobrepasa. El padding es importante para luego poder generar LODs de los modelos y que no haya sangrado en las texturas.
5. Colocar el prop junto a otro ya finalizado que emplee las mismas texturas para corregir incongruencias visibles en el texel density que no se hayan visto previamente en el escalado manual.

Resolución de Textura	Texel Density recomendado (px/m)	Uso
512 × 512	128–256 px/m	Props muy pequeños o lejanos
1024 × 1024 (1K)	256–512 px/m	Props pequeños, elementos secundarios
2048 × 2048 (2K)	512–1024 px/m	Props medianos o importantes en primer plano
4096 × 4096 (4K)	1024–2048 px/m	Props grandes, arquitectura, personajes principales
8192 × 8192 (8K)	2048+ px/m	Assets enormes en cine o escenas cinematográficas

Tabla 19. *Texel densities recomendadas por resolución de imagen, según Epic Games en Unreal Engine Documentation (2023).*

En algunas ocasiones se ha optado por añadir geometría extra que no aporta a la silueta para controlar mejor las islas de UVs. En el siguiente ejemplo (Fig.) se ve cómo hay cortes a mitad de una balda en una estantería cuando no tienen sentido en el modelado en sí, pero una vez viendo la intención del texturizado se entiende bien el porqué. Aunque un buen unwrap suele configurar bien las islas, si se necesita ajustar a unas medidas concretas y el stretching es más alto de lo asumible, es mejor añadir un corte de geometría en el lugar en cuestión para controlarlo directamente. Aquí el nuevo corte permite disponer mejor la separación y posición de los dos tablones que componen la balda.

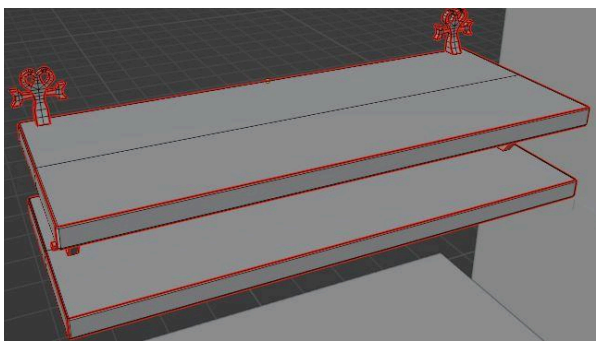


Figura 47. *Geometría adicional para manejo de UVs.*

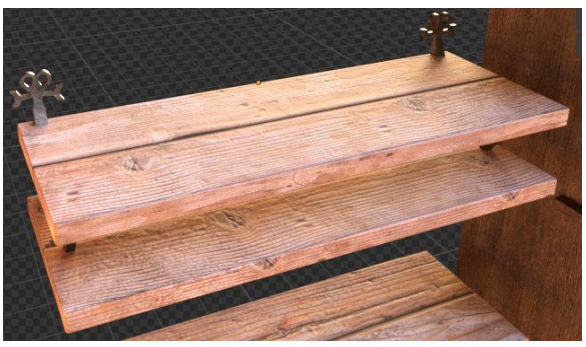


Figura 48. *Geometría adicional renderizada.*

EXTRA: CUANDO SÓLO SE TIENE EL ALBEDO

Al texturizar mediante tileables se diseñan las texturas para assets genéricos, no para una geometría concreta, por lo que no hay detalle real de profundidad más allá de la que se genera en el propio material, pero no para el modelo.

Para texturizar el suelo de la taberna era necesaria una textura tileable de tablones con sus hendiduras, además debía usarse la misma madera que se usaba para el resto de la construcción. El problema era que no había geometría real que apoyara esas hendiduras de las tablas. Exportar un plano con la profundidad modelada como para hacer un trim sheet sirve si se usa un patrón y distribución tileable, pero el relieve conseguido sin recurrir a bakear mapas que impiden el tileado como el Ambient Oclussion o World Space Normal, no era suficiente como para resaltar la separación entre los tablones. El resultado era demasiado plano.

Como solución se utilizó “Materialize”, un software creado en Unity y que se ha utilizado ya en proyectos importantes dentro de la industria, como el remaster de los videojuegos de Uncharted. Este programa permite generar el resto de mapas de texturizado a raíz solo del Albedo o Diffuse. No es necesario nada más. El resultado no es tan espectacular como podría serlo si se bakean AO y curvature reales en base a un modelo concreto, pero es sorprendente dadas las limitaciones.

Lo primero es colocar Materialize y las texturas que se vayan a importar en un lugar en el que no le sea necesario pedir permisos de almacenamiento para leer y escribir archivos temporales (evitar el escritorio), o de lo contrario no se cargarán en el programa.

Después, se hace clic en el botón “O” (open) de la casilla para el Diffuse. Una vez se carga este mapa se puede empezar a crear el resto. Ahora se puede crear el Height Map desde su botón Create, que ya aparece habilitado. A raíz del Height se puede crear después el Normal. Luego viene el Metallic y después el Smothness (roughness invertido para Unity). Por último, AO y mapas auxiliares. Cada una de las creaciones de mapa tiene sus opciones para customizarlo, como la cantidad de contraste en la imagen generada, de detalle “crisp” en las normales, etc.

Para exportar las texturas es necesario darle a “Save”, que además de guardar un archivo de programa a partir del que se puede retomar el proyecto, también exporta las texturas a la dirección de guardado.

Debajo puede verse cómo se creó el material en cuestión, el que generó el problema de solo tener el albedo para empezar a construir: el suelo tileable de tablas de la taberna.

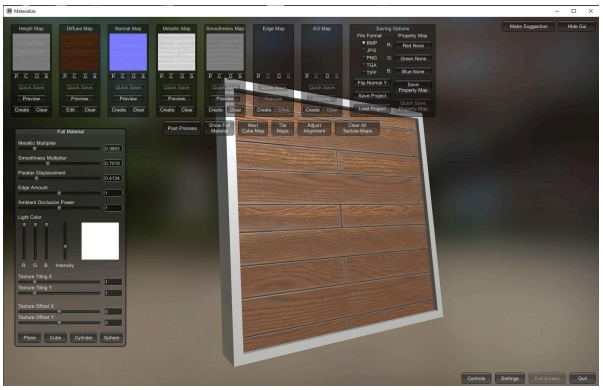


Figura 49. *Creando un material con Materialize.*

Figura 50. *El material resultante renderizado en Blender.*

3.5.3.4 **Texturizar props pequeños:** atlas de texturas

Este paquete, a pesar de ser lo más óptimo en draw calls, al final utiliza texturas separadas para cada prop en lugar de atlas combinados; por pequeños que sean. Esta decisión se ha tomado con el objetivo de facilitar la edición individual de materiales por parte del cliente, mejorar la compatibilidad con sistemas de LODs automáticos y evitar posibles artefactos de interpolación o sangrado. Además, el uso de texturas pequeñas y optimizadas permite una carga eficiente en motores con soporte de streaming por asset. En futuras versiones del paquete se considerará una versión con atlas unificado para proyectos que prioricen draw calls mínimos.

3.5.3.5 IA en Texturizado

Meshy AI ha resultado ser especialmente buena en texturizado, ya que admite archivos 3D como input y los texturiza en función de las indicaciones que se le den. Puede generar UVs poco lógicas, pero si se le dan hechas, genera un producto totalmente usable. Apenas requiere de limpieza manual posterior. Tras varios intentos, se ha verificado que el resultado es mucho mejor cuando la intención es crear assets de estética visual estilizada; para realismo no es tan bueno porque, lógicamente, es lo más difícil.

Entre el modelo económico en base a créditos que usa Meshy en lugar de una suscripción fija, y a que hay opciones mejores para conseguir texturas, se ha decidido no utilizar Meshy para el producto final, por lo que no se adjuntan capturas. Eso sí, tras usar esta IA cabe resaltar que recibir un modelo 3D con las texturas ya puestas es un punto muy positivo.

La IA escogida como ayuda para el texturizado, como ya se ha comentado en la metodología, ha sido ChatGPT. La comprensión superior que tiene para analizar lo que el usuario quiere y su posibilidad de traducirlo a un prompt que entienda su IA generativa de imágenes (DALL-E 3) es lo que coloca a ChatGPT como la mejor opción viable a fecha de mayo de 2025 para el texturizado. Si bien no puede recibir modelos 3D y devolverlos texturizados

como sí que puede hacer Meshy AI, ChatGPT permite dar descripciones muy detalladas como input, además de subir imágenes de referencia de lo que se pretende obtener.

Los resultados obtenidos son de una calidad sin precedentes y de la autoría de quien los genera, pero como funcionan en base a prompts y no en base a una rotoscopia real que mida las distancias entre los puntos clave, ChatGPT no es capaz de replicar con exactitud una misma imagen. Esto es un punto clave en el texturizado porque para generar el resto de mapas, el encaje entre ellos ha de ser exacto. La IA trata de replicarlos reutilizando el mismo prompt y las ecuaciones que ha extraido para llevarlo a cabo, pero el resultado no es idéntico.

Entonces, ChatGPT es muy buen generador de mapas Albedo/Color/Diffuse, pero no se recomienda para generar el resto de mapas si el modelo que va a utilizar la textura es protagonista o va a estar instanciado a un buen tamaño. En cambio, si se trata de elementos pequeños, la fidelidad percibida entre los mapas cuando se generan todos con ChatGPT es suficiente al ojo. Por consiguiente, se recomienda encarecidamente su uso para ahorrar tiempo. Si se requiere precisión, la solución es generar el resto de mapas en Materialize.

Los resultados obtenidos han sido casi instantáneos (1 o 2 generaciones) cuando se pidieron texturas tileables genéricas reforzadas por una imagen de referencia. En cambio, cuando se pidió algo menos universal que por ejemplo, un muro de mampostería, el resultado sí que se tardó bastante más en obtener de manera satisfactoria.

En cuanto a correcciones, el resultado es bueno y rápido cuando se piden cambios de color. Por el contrario, es sustancialmente peor cuando se piden cambios de distribución. Por ejemplo, ChatGPT aún tiene problemas para mover elementos a otros lugares de la imagen o para cambiarlos de color. En estos casos, muchas veces lo que hace es estirar la imagen hacia la dirección o tamaño nuevos indicados, generando franjas que parten la textura. Aún cuando se le indica el error que ha cometido y lo que se pretende que realmente haga, responde con un bias de confirmación para indicar que el usuario tiene razón y que se ha subsanado el error, pero luego resulta que no es verdad. En estos casos cuando se obceca en repetir imágenes con el mismo problema, la solución es pedir la generación en un chat nuevo y adaptando el prompt para evitar acabar en el mismo bucle una vez más.

Figura 51. *Imagen de referencia para ChatGPT*

Figura 52. *Textura resultante de ChatGPT.*

En las figuras referidas arriba, Fig. y Fig, , se expone la referencia de estilo que se le suministró a ChatGPT y el resultado obtenido. El prompt empleado fue el siguiente: “Genera una textura tileable en todos sus ejes, de suelo de mampostería 2048x2048 en base a esta referencia. Ten en cuenta que ha de permitir un tileo fuerte a gran escala, por lo que evita que se vea la repetición del tiling dentro de lo posible. Añádele un tinte más anaranjado”.

Es importante especificar que el tileo ha de ser en todos los ejes o es posible que solo lo haga en uno de ellos, cosa que luego es difícil de corregir y puede requerir de abrir un nuevo chat. También parece entender mejor la resolución de imagen si se le da numéricamente que si se le indica, por ejemplo, como “2K”. En este último caso, podría contestar en texto que ha generado una imagen de esa resolución, pero luego al descargarla viene en 1k, que es el tamaño al que chatGPT genera texturas por defecto.

También es importante recalcar que las texturas han de descargarse en un período corto de tiempo o podrían desaparecer por cómo OpenAI, la empresa matriz de ChatGPT, gestiona los tokens temporales de descarga. En tan solo 15 o 30 minutos el botón de descarga podría expirar por lo que alegan ser motivos de seguridad, aún cuando la imagen sigue siendo visible en el chat. Una nueva generación de “la misma imagen” no lo es realmente, ya que reinterpreta el prompt. En casos normales, las descargas se mantienen disponibles durante varias horas o incluso días, pero lo otros también ocurre con relativa frecuencia. En un caso de necesidad, se podría hacer una captura de pantalla y rescalar la textura en PhotoShop a su tamaño intencionado.

ChatGPT se ha usado como base para generar 2 trimsheets y 3 tileables en este proyecto, que son las siguientes: *stone_floor, architecture_stone, trims_cluttler, trims_glazed_clay y mossy_stone.*

	Props Pequeños	Props Grandes
Generar Albedo	Sí	Sí
Generar el resto de mapas	Sí	No, usar Materialize en base al Albedo generado.
Generar Height Maps	Sí	Sí, pero luego modificarlo a mano desenfocando mucho y aumentando el contraste (funciona mejor para POM y Displacement).
Generar Trimsheets	Ver derecha->	Con cuidado, puede romper tiling y es difícil modificar las proporciones entre materiales.

Tabla 20. *Cuándo usar ChatGPT para texturizar.*

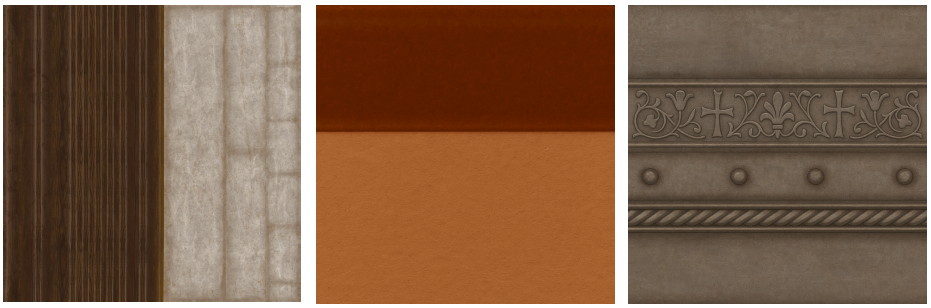
3.5.4 Sorteando limitaciones y añadiendo personalización

3.5.4.1 Trim Sheets

Se han desarrollado tres trimsheets de cara a este proyecto, una 4K para los elementos arquitectónicos, otra 1K para props pequeños de cubertería, y una 512 x 512 para recipientes de arcilla para beber. Se han elegido estos props porque eran los que se identificaron como buenos beneficiarios de detalle extra, pero que aún así podían permitirse cierto tileado.

El método utilizado fue el descrito en 3.3 Metodología, donde se incurre en la importación en Substance Painter de mallas cuadrangulares con cierto relieve en zonas planeadas. Así se puede texturizar por capas una textura genérica y aprovechable para assets de diferente forma y tamaño, pero también se puede aprovechar el uso de máscaras inteligentes que leen los recovecos del modelo para crear desgastes.

La trimsheets de edificios, trims_woodPlaster.png, no es tileable, pero permite reutilizar partes detalladas como tablones o partes de pared que simulan ambient occlusion pintado. Las trimsheets pequeñas, se usaron para detallar las copas, jarras y cálices. Estas sí que son tileable en su eje horizontal y permiten, gracias a ello, repetir cenefas decorativas y zonas planas de metal sin ornamentos.



Figuras 53-.55
Trimsheets del proyecto

3.5.4.2 Card decals

De cara a que el paquete alcance al mayor público objetivo, emplear decals para los detalles se determinó como poco viable por la falta de soporte en algunos motores. Como solución se optó por utilizar una estrategia mucho menos conocida que se denomina “card decal”.

Un card decal cumple la misma función que un decal estándar: añadir detalles únicos y pequeños que por su tamaño no requieran de una extrusión real en la geometría del modelo. La diferencia es que en lugar de ir proyectado desde un sistema volumétrico a superficies, usa geometría.

Aunque visualmente no existe diferencia con el decal al uso, su funcionamiento real es totalmente diferente, y es que es idéntico al de un material normal y corriente: una serie de texturas aplicadas a una geometría. Consisten en aplicar formas pequeñas, sencillas y muy finas, normalmente planos con transparencia, pegadas, o ligeramente separadas, a la malla del

modelo que se está texturizando para darle una capa de detalle extra sobre el material principal. Se pueden usar para añadir arañazos, desgastes, suciedad, remarcar hendiduras o salientes del modelo que no vienen contemplados en la textura del material principal, añadir tornillos, clavos, calcomanías de iconos o texto, etc.

Si bien al llevar geometría real, y por tanto, conllevar la creación de materiales extra, el motor genera más draw calls, la carga nueva de geometría es muy poco apreciable, y si se reutiliza el mismo card decal tampoco se generan tantas draw calls extra. La ventaja principal es la compatibilidad, y es que un card decal funciona en todos los motores sin soporte adicional. Un decal normal proyectado solo se puede aplicar en Unity si se usa el render pipeline HDRP, que es poco utilizado por la incompatibilidad con la mayoría de assets de la tienda y por la dificultad extra que requiere de optimización. Por el contrario, Unreal Engine 5 ya incorpora decals proyectados en todas sus versiones.

Los card decals también se pueden exportar directamente desde el programa de 3D dentro del modelo sin que sea el cliente el que tenga que colocarlos luego en el motor, como ocurriría con los decals normales. Aún así, es preferible no saturar un mismo modelo con muchos card decals diferentes porque no es buena idea tener más de 3 o 4 materiales por prop. Aunque se reutilicen esos materiales, sigue sin ser lo más óptimo.

Lo que a nivel personal se recomienda es utilizar entre uno y tres materiales normales para el texturizado general del asset (preferiblemente dos), un card decal para marcar zonas oscuras y otro para añadir detalles únicos. Más de eso es “overkill”. Si fuera estrictamente necesario, se podría contemplar otro card para marcar salientes o zonas claras, o incluso también un segundo detalle.

Para aplicarlos, se crea un plano dentro del propio modelo, nunca como un objeto aparte. Este plano se escala al tamaño que vaya a tener el detalle y se coloca todo lo cerca que se pueda de la malla del modelo, pero sin que la atraviese. Es decir, que siga siendo visible, pero por poco. De cara a que pueda verse el card desde lejos, es recomendable que no vaya pegado exactamente en la misma posición que ocupa la malla original, puesto que podría no verse conforme la cámara se aleje del modelo, o incluso generar artefactos de iluminación. Es mejor dejar cierto margen de separación para que no ocupe el mismo espacio, pero que esté muy cerca. Si la superficie sobre la que aplicar está curvada será necesario añadirle algunos cortes más al plano del card para poder doblarlo alrededor de la geometría. Cuanta menos geometría se añada, mejor, pero ha de ir pegado a la superficie del modelo sin atravesarlo o quedar descolgado. Luego, se añade un material más al modelo, que es el que lleva los detalles con transparencia a añadir, y se le aplica al card. Por último, se ajusta tal y cómo se vea conveniente.

En la Fig.21 se aprecia cómo van colocados los cards: uno se repite y es un textura de un clavo y el otro es un degradado con poca opacidad de negro a vacío para falsear AO y que parezca que está sombreado. En la Fig.22 se ve el modelo renderizado y se comprueba que visualmente cumple la misma función que un decal.

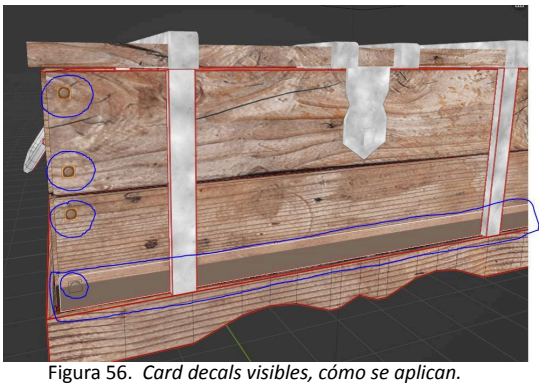


Figura 56. *Card decals visibles, cómo se aplican.*

Figura 57. *Card decals en el producto final.*

La siguiente tabla resume las diferencias entre usar un card decal y uno proyectado:

Característica	**Card Decal**	**Decal Real (Proyectado)**
Geometría	Plano físico añadido (extra mesh).	No añade geometría visible, se proyecta en tiempo real.
Aplicación	Se exporta desde el programa de 3D; se vende ya colocado.	Requiere aplicación manual por el cliente después en el motor.
Precisión	Depende del alineamiento manual con la malla.	Se adapta perfectamente a la geometría subyacente.
Draw Calls	Aumenta los draw calls (nuevo mesh y material independiente por cada card)	También aumenta draw calls, pero de forma más eficiente en muchos motores modernos.
Compatibilidad	Máxima compatibilidad.	Requiere que el motor soporte proyección decal (Unity HDRP, UE5, etc.)

Tabla 21. *Diferencias entre un decal proyectado y un card decal.*

3.5.4.3 Máscaras de color

Las máscaras de color han funcionado tal y cómo se esperaba. Para el proyecto se han hecho máscaras siempre y cuando un simple cambio global de color no solucionara las necesidades de personalización. En casos donde se consideró alta la probabilidad de que el cliente pudiera querer cambiar solo una parte concreta de la textura de color, pero no el resto, es en los que se ha hecho. Se han implementado entonces máscaras para cambiar:

- Cenefas decorativas en la porcelana.
- Bordados en los manteles.

Para ver cómo implementar estas máscaras en materiales preparados para que el cliente las use fácilmente en los motores gráficos, ver 3.5.5 Empaquetado en motores.

3.5.4.4 Texel Density: trucos y engaños cuando no es posible igualarlo

Uno de los principales problemas encontrados a la hora de texturizar assets que tienen tamaños muy diferentes y aún así deben compartir texturas para unificar la temática visual del paquete, ha sido la inconsistencia en el texel density. Por ello, como rehacer el material a mayor resolución consume mucho tiempo además de espacio extra considerable en disco, se ha tratado de engañar al ojo haciendo que una textura con texel density bastante menor parezca que tiene uno mucho mayor.

Como ejemplo, se ha tenido que reutilizar el material de madera que utilizan las partes arquitectónicas, que son assets grandes, para hacer versiones alternativas de los props pequeños, puesto que muchos de ellos usan maderas y tonos muy diferentes. Entonces, si se quería unificar visualmente la taberna con sus props, era mandatorio que al menos una versión de ellos diera la sensación de utilizar la misma madera y colores en su concepción. El problema era que los props usaban mapas detallados a resolución 2K, y que la madera de los edificios empleaba resolución 4K en una trim sheet que englobaba varios materiales (no solo madera), por lo que al utilizar un tablón de la trim sheet para texturizar una zona que en las versiones de 2k ocupaban prácticamente todo el espacio UV, resultaba en una resolución percibida como ridículamente baja en la trim sheet.

Entonces, se procedió a pasar el mapa albedo, conocido también como diffuse o color, y el mapa de normales por un filtro de enfocado fuerte (sharp) en PhotoShop. Esto, acompañado de un aumento de ruido en el mapa roughness, consigue una resolución percibida mayor a pesar de que realmente no lo sea. Ahora, ambos materiales aplicados al mismo prop tenían consistencia. Seguía habiendo cierta diferencia, pero solo observando en detalle al colocarse muy cerca del prop, a simple vista no es perceptible. El resultado al ojo se calcula como abarcable dentro del 15% de diferencia máxima recomendada entre props para el texel density.

Eso sí, es recomendable que la nueva versión enfocada no reemplace a la original, puesto que entonces los assets que lo usaran con anterioridad ahora van a verse más nítidos también, generando otra inconsistencia. Los nuevos mapas han de constituir una nueva versión usada solo en los props que requieran mayor detalle.

Para aumentar texel density percibido:

1. Enfocar (sharp) los mapas Albedo y Normal.
2. Añadir ruido y más detalle al mapa Roughness.

3.5.4.5 UVs y lightmaps

A efectos prácticos un lightmap es lo mismo que bakear los mapas a la textura en programas como Substance Painter, pero además lo hace para para es prop con respecto de la escena. Es decir, no solo se aplica sobre el propio asset, sino también sobre la sombra que proyecta sobre la pared de detrás y el suelo; con el resto de la escena.

Como ya se ha discutido con anterioridad, los assets texturizados por overlap no pueden llevar información adicional sobre la textura, entonces, la solución es separar en dos mapas de UVs la funcionalidad del texturizado de la del bakeado. Esto impide detalles de iluminación en el asset a la hora de crearlo, pero permite que se repliquen después en el motor gráfico.

En este proyecto, todos los props que se han texturizado por overlap lo llevan en el canal principal (slot 0) y, además, llevan un mapa adicional en el canal secundario (slot 1).

Aunque aquí no haya sido el caso, existen ocasiones muy concretas en las que ambos canales de UVs pueden llevar overlap y no es posible calcularles los mapas de luz en el motor. Se ha investigado y resulta que es posible tener un AO bakeado en el lightmap sin falta de recurrir a calcularlo en tiempo real con SSAO o RTAO en un asset que tiene overlap. Es decir, hacer lo mismo, pero sin el canal secundario de UVs limpio. El proceso es el siguiente:

1. Se duplica el modelo.
2. Se le sustituyen las UVs con overlap a la copia por unas limpias.
3. Se oculta el prop con overlap en la escena del motor gráfico y se coloca en su lugar exacto la copia sin overlap.
4. Se bakea en el motor usando el prop sin overlap solo para conseguir el lightmap.
5. Se elimina la versión sin overlap y se vuelve a mostrar el prop original
6. Ahora el prop texturizado por overlap coincide en el espacio con la iluminación correcta que debería tener.

Se trata de un flujo funcional, pero que añade complejidad extra al proceso de creación de assets. Solo se recomienda entonces para casos extremadamente específicos en los que sea necesario por motivos técnicos. Por ejemplo, para props muy importantes y con protagonismo en cinemáticas o que van asociados al controlador de personaje; y siempre que ambos canales de UVs ya tengan overlap y no sea posible rehacerlos.

3.5.5 **Empaquetado en motores:** Unreal Engine y Unity

NOMENCLATURA

Lo primero antes de exportar a motores es renombrar todos los archivos y texturas con un nombre coherente que facilite encontrarlos después. También un almacenamiento estructurado es mandatorio para una más rápida importación en otros programas. La nomenclatura cambiará después en cada motor, pero así ya se tiene una base. Se recomienda usar nombres en inglés y separar las palabras que no hagan referencia a la misma cosa con un "_" y las que sí atañen a la misma cosa, con una mayúscula intercalada al principio de la

palabra. Ejemplo: “roughWood_normal.png”. Esto se conoce como “camel case” y está muy extendido en el ámbito de la programación y la tecnología en general. A veces, algunos paquetes investigados han resultado utilizar “snake case” exclusivamente para las texturas, que es lo mismo que camel case, pero en lugar de juntar palabras que hacen referencia a lo mismo en un único espacio entre “_” y marcar el inicio de la segunda palabra con mayúscula, directamente las separan con el guion bajo para evitar usar mayúsculas. Como conclusión, es preferible el camel case porque casa mejor con lo que se espera en los motores. Aquí debajo se expone la nomenclatura comprobada para empaquetar en motores (ver Tabla).

Como puede observarse, Unreal fuerza unos prefijos por convenio que en Unity son opcionales. Si el objetivo son ambas plataformas, se puede optar por nombrarlo todo una única vez en formato Unreal, pero tampoco es algo recomendable ya que existen algunas diferencias entre cómo ambos motores llaman a las cosas. Lo mejor es nombrar todo lo común en formato Unreal, y a lo no común, que son las mallas de los modelos, no añadirle prefijos. Esto es perfectamente importable en Unity, y luego se pueden añadir los prefijos restantes manualmente en Unreal: “SM” (Static Mesh).

Elemento	**Unity (URP/Built-in/HDRP)**	**Unreal Engine (UE4/UE5)**
Modelo 3D	NombreAsset.fbx	SM_NombreAsset.fbx (SM = Static Mesh)
Prefabs / Blueprints	NombreAsset.prefab	BP_NombreAsset
Material	M_NombreMaterial.mat o NombreMaterial.mat	M_NombreMaterial (prefijo obligatorio)
Textura Albedo/BaseColor	NombreAsset_Albedo o NombreAsset_BaseColor	T_NombreAsset_BaseColor (T = Texture)
Escena	No hay estándar explícito	L_NombreMapa (L=level)

Tabla 22. *Nomenclaturas de Unity y Unreal.*

PIVOTES DE OBJETO

Otro aspecto importante al tratarse de asset para videojuegos es el centro del objeto, y es que para colocar assets en una escena 3D, ya sea en Unity, Unreal o en el mismo Blender, es más fácil hacerlo si los orígenes del modelo están en lugares lógicos. Por norma general, el origen ha de colocarse en el centro de geometría de la base de un objeto, a sus pies. También hay excepciones, como assets que normalmente se colocan iterativamente en contacto con otros, o que se suelen colocar en torno a un punto de pivote, como pueden ser las vallas, la tapa de un cofre o una ventana. En estos casos es preferible poner el centro del objeto en aquel punto por donde entraría en contacto con la siguiente colocación de un asset del mismo tipo, o por donde interesa que pivote al rotar.

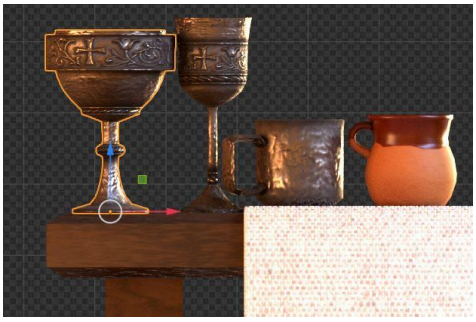


Figura 58. *Centro de objeto estándar.*

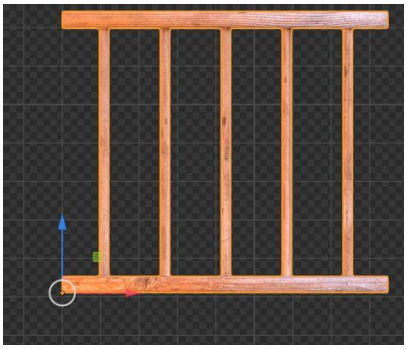


Figura 59. *Centro de objeto iterable, en su punto de contacto.*

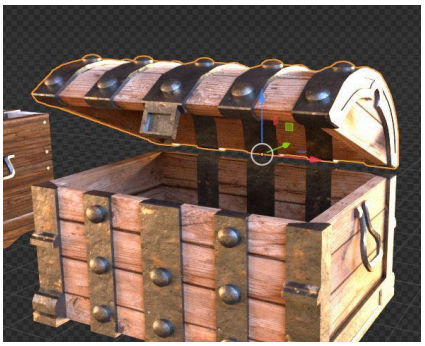


Figura 60. *Centro de objeto en su punto de pivote idóneo para rotación.*

Dejar los pivotes en su lugar original en lugar de modificarlos para facilitar una mejor colocación del asset en una escena 3D, no solo es una mala práctica, sino que también es motivo de rechazo en los portales de venta para Unity y Unreal. Una “colocación centrada y en la base, o en su punto lógico para assets modulares” es mandatorio para publicar según las Submission Guidelines de Fab y Unity Asset Store.

OPCIONES COMUNES DE EXPORTACIÓN

Si se activa la opción “Tangent Space” en el apartado “Geometry” al exportar un modelo a FBX, Blender avisará de si hay Ngons en la malla, cosa que es mucho más rápido que hacer selecciones por tipo de polígono a todos los props uno por uno. Como se verá más adelante, usar Tangent Space es totalmente compatible con los flujos de exportación a Unity y, especialmente, a Unreal. Para remediar los NGons, lo mejor es aplicar un modificador “Triangulate” y volver a exportar. Si vuelve a dar error, entonces sí que habrá que entrar en

modo edición del objeto, ir a Select->Select All by Trait->Faces by Sides, y seleccionar “Greater than” y “4” en el menú de contexto, para luego solucionar manualmente los NGons que aparecerán seleccionados en la malla. Si la cara seleccionada parece no tener NGons, es que tiene vértices duplicados, que se pueden eliminar en modo wireframe seleccionando cada vértice arrastrando con “Select Box” y después “Merge At Center” (Ctrl + M). No hay ningún problema con triangular el modelo, puesto que Unity y Unreal van a triangular la malla igualmente al importar; los quads no se van a mantener de todas formas.

3.5.5.1 Unity 6

3.5.5.1.1 Exportación de Blender para Unity

El proceso que ha resultado en una correcta orientación de los modelos al importarse es el siguiente:

1. Colocar el modelo en el origen de coordenadas, es decir, en la posición (0,0,0). De lo contrario, habrá un desfase no deseado entre el origen del modelo y su posición relativa en la escena. En palabras llanas: se estaría moviendo un objeto “invisible” porque su cuerpo real está muy lejos.
2. Rotar el modelo hasta que tenga la dirección deseada mirando al frente, que en Blender es -Y.
3. Aplicar todas las transformaciones, posición incluida. Ctrl + A para abrir el menú de contexto de aplicación de transformaciones. Después, pulsar “Apply all transforms”. Esto hará que el objeto tenga una transformación limpia aplicando todas las modificaciones previas, lo que quiere decir que si había una rotación de 90º, ahora la información del objeto dirá que no tiene rotación alguna, pero el objeto seguirá rotado 90º. Esto le “aplica” la rotación como nuevo punto estándar de partida. Ocurre lo mismo para la posición y la escala.
4. Para exportar el archivo se irá a File-> Export-> .fbx.
5. En el menú de exportación de Fbx es necesario aplicar meticulosamente las opciones tal y cómo se ven en la Fig., con apply scalings en modo “FBX All” y “use space transform” desactivado. Esta configuración de ejes “+Y forward” y “+Z up” es tal y cómo ha de leerla Unity. Sí, no es exactamente el convenio de Blender, el eje profundidad (Y) va en positivo en lugar de en negativo. Esto resuelve la ambigüedad ya comentada entre ejes. Además, hay un par de cuestiones a configurar en el menú que no aparecen en la figura, que son marcar “Export Selected Only” para limitar la exportación únicamente al asset, y en el apartado “Geometry” asegurar que el campo dice “Normals Only”. Esto último asegura una correcta exportación de las normales específicas de Blender a un estándar que entiendan el resto de programas (las traduce a “smoothing groups”).
6. Lo siguiente es copiar el nombre del asset con F2 (renombrar) en la jerarquía y Ctrl + C, para luego pegarlo en el nombre de destino para la exportación (Ctrl +

V). Por defecto, el nombre es el del archivo .blend, no el del objeto seleccionado.

7. Por último, se arrastra el archivo a la carpeta del proyecto en Unity para importarlo. Luego, se hará clic en el fbx que se acaba de cargar para que aparezca su menú de contexto en el editor, donde se irá a la pestaña “Model” y se marcará “Bake Axis Conversion” (ver Fig.). Después, se le da a “Apply”. Esta configuración le indica a Unity que ha de sobreescribir su propia conversión de ejes con la información que lleva el fbx a nivel interno. Ahora, el modelo estará bien orientado, tanto en la ventana de visualización como en la escena cuando se instancia.

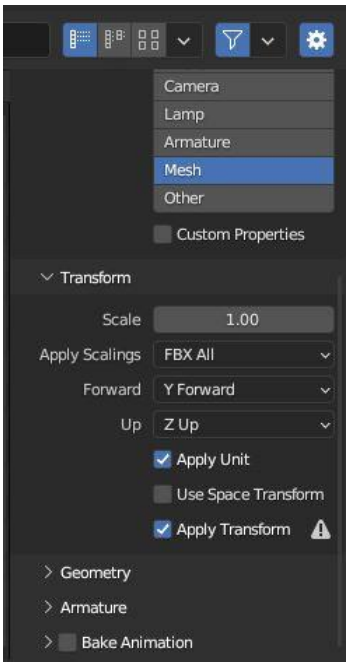


Figura 61. *Exportación Blender-Unity.*

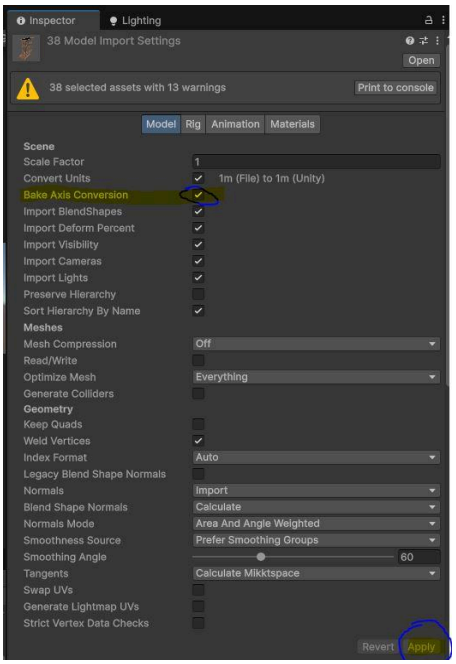
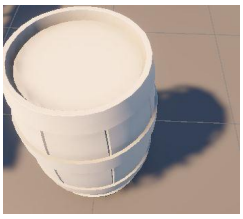
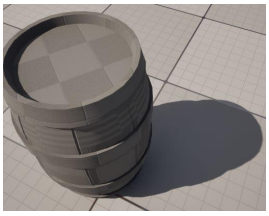


Figura 62. *Importación en Unity.*

En el anexo se adjunta un vídeo que se ha grabado expresamente para explicar este proceso. Explica los conceptos teóricos y los pasos prácticos.

Cabe resaltar que en Unity, efectivamente la geometría flotante puede generar fugas de luz y artefactos en las sombras, aún cuando el prop está bien hecho y los vértices sin coser están bien planteados y colocados, pero se soluciona al bakear. Solo ocurre con iluminación en tiempo real. Se puede solucionar bajando el normal bias a valores cercanos a 0, pero no se ha de poner del todo a 0 o se generarán artefactos nuevos. No obstante, esta opción se encuentra en las opciones de cada una de las luces de la escena, pero no se puede controlar en los assets del paquete, por lo que corre a cuenta del cliente. Eso sí, es mejor documentarlo en los archivos asociados de tipo readme.txt que se recomiendan incluir encarecidamente en el producto desde Unity Asset Store.

Estas fugas de luz, en cambio, en Unreal no se producen, porque el sistema de iluminación en tiempo real es más potente e interpreta correctamente la geometría que está muy cerca o en la misma posición como un único objeto.

Figuras 63 y 64. *Huecos en sombras por geometría flotante en Unity (izquierda) vs Unreal (derecha).*

3.5.5.1.2 Materiales en Unity: Smoothness y Metallic empaquetado

Resulta que Unity entiende de forma diferente el mapa "Metallic". Además de contener la información que normalmente llevaría este mapa, en Unity se combina como mínimo el Roughness (invertido), y opcionalmente con el Ambient Occlusion. El concepto de textura metálica es bastante diferente y no ha de confundirse con el estándar. Este flujo combinado se denomina comúnmente PBR Metallic-Smoothness.

Al ser mapas en escala de valor, es decir, puntuaciones de blanco y negro, se pueden representar con un único número los colores a representar en cada píxel, de ahí que se pueda empaquetar cada mapa en un único canal del RGB. No hace falta combinar 3 colores, con uno solo basta. Unity combina hasta 3 mapas de la siguiente manera:

Canal	Mapa Empaquetado
R (rojo)	Metallic
G (verde)	Ambient Occlusion (opcional)
B (azul)	Libre o ignorado
A (alfa)	Smoothness

Tabla 23. *Empaquetado de mapas en Unity.*

CREACIÓN DE MAPAS

El mapa que en cualquier otro entorno se entiende como Metallic va en el canal Rojo. En el verde, por convenio, es donde se debería de meter el AO, pero no solo es opcional, sino que Unity no lo va a leer de ahí por defecto, salvo que se le especifique creando un shader personalizado o modificando uno existente; no va a funcionar en URP Lit de primeras, lo interpretaría como vacío y no lo mostraría. El mapa azul se deja vacío siempre. En el canal alfa va el mapa Roughness, pero invertido, que se conoce en Unity como Smoothness.

Para combinar los mapas en estándar Metallic-Smoothness, en Photoshop:

1. Abrir un archivo nuevo en modo RGB (32 bits), nunca en escala de grises. Este va a ser el archivo principal.
2. Abrir los mapas Metallic y Roughness como nuevos archivos independientes entre sí y del archivo principal (arrastrar cada imagen como ventana nueva hasta tener tres proyectos abiertos: el principal y los otros dos)
3. Copiar el mapa Metallic (Ctrl + A para seleccionarlo todo, y luego Ctrl + C para copiar), y pegarlo en el canal rojo con Ctrl + V (usar la pestaña de canales, no la de capas). Es importante asegurarse de que solo se pega en el mapa rojo y no en el RGB completo o en otro canal. En caso de no tener mapa metálico específico, se pinta todo el mapa de blanco puro si el objeto a texturizar se entiende como metálico, o de negro puro si no es un metal.
4. Añadir un canal extra. El nombre no importa. Photoshop lo interpreta como alfa por la posición que ocupa en el orden de los canales.
5. Pegar el mapa Roughness invertido (Ctrl + I), que es el Smoothness, en el canal alfa.
6. Los canales que no se usan hay que dejarlos o bien en negro puro o en blanco puro, ya que existir, existen. Aunque se desmarquen sus casillas, estas van a seguir yendo empaquetadas en la imagen, pero ocuparán menos si son blanco o negro puros. Salvo excepciones, se prefiere el blanco puro, ya que este color no se multiplica en modo de fusión con otros colores.
7. Guardar como TGA para mantener un canal alfa intacto aunque pese más el archivo. En versiones recientes de PhotoShop, TGA ya no aparece como una opción en Exportar, hay que ir a *archivo-> guardar una copia*. La exportación en PNG tiende a dar problemas para mantener información en el canal alfa, suele interpretarse solo la transparencia; mejor TGA.

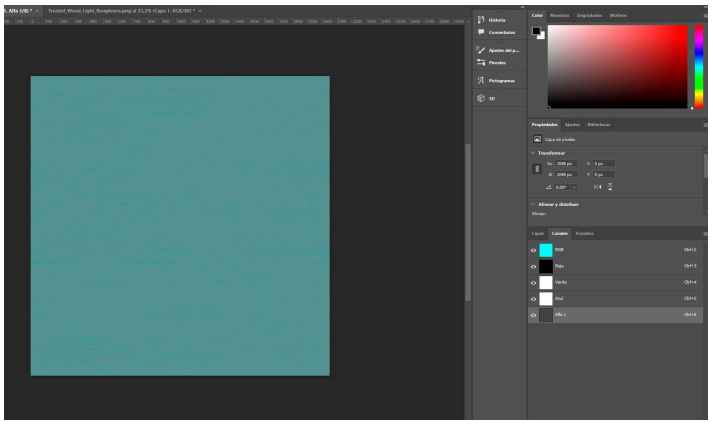


Figura 65. *Ejemplo de distribución Metallic-Smoothness en Photoshop.*

El resto de mapas de texturizado en Unity son idénticos a los estándar en la industria y es preferible que vayan en PNG para que ocupen menos, pero jamás en formato JPG. Incluso el albedo debe ir en PNG. JPG está prohibido en las tiendas a no ser que se justifique como referencia o UI no interactiva, puesto que implica degradación progresiva; cada vez que se reexporta pierde calidad y además puede ver alterado su color base por su tipo de compresión.

SETUP DE MATERIALES

Conviene comprobar que los mapas de normales no están invertidos si se han exportado para otra API gráfica diferente a la de Unity. Para ello se va al archivo original de la textura en la carpeta del proyecto de Unity (donde se haya cargado) y se marca “Invert Y”. Tras darle a “Apply”, se puede ver si la visualización mejora o empeora.

Unity no visualiza los mapas técnicos tan mal como Unreal si no se le indica que lo son, pero es preferible hacerlo si se quiere mayor fidelidad de material. Para indicarle que son técnicos y que no son un Albedo, se desmarca la opción “sRGB” de la textura en su menú de contexto del editor (cuando se le hace clic). Es similar a indicar que la textura es “Non-Color” en Blender. No obstante, se ha comprobado que la diferencia al no indicar el tipo de mapa no es muy acusada en Unity, a no ser que se trate de un mapa empaquetado o de un mapa de normales, que sí que requieren desmarcar sRGB para una correcta visualización.

Por último, los mapas de normales han de ser especificados como tales marcando “Is Normal Map” en la textura, pero Unity lo hace automáticamente al cargarlos a un material. Un pop-up salta advirtiendo de ello, por lo que es difícil de olvidar.

Como punto positivo, Unity permite por defecto aumentar o reducir la influencia del Smoothness con un slider en las opciones de material, a pesar de ya contar con una textura para ello. En Unreal habría que programarlo desde el shader. Este slider también lo tiene para el Normal Map.

La creación de los materiales en sí es muy sencilla, basta con seleccionar las texturas correspondientes desde los slots reservados para cada mapa en el material por defecto de Unity 6, que se crea en formato Metallic-Smoothness con el shader URP Lit.

Si se pretende hacer materiales complejos como transparencias o relieves fuertes y realistas con“Parallax Occlusion Mapping”[32] o con “Displacement”[33], no se puede hacer con shaders por defecto de manera visual. Tampoco es posible hacer máscaras de color. La solución obvia es usar Shader Graph, un plugin de Unity muy potente, pero que se considera una dependencia de paquete, es decir, requiere de una importación extra para que el asset funcione. No es funcional por sí solo. Esto lo prohíbe Unity Asset Store. Entonces, para emplear estos nodos avanzados que consiguen materiales de una calidad superior en un asset que se va a vender, la única opción es programar el shader a mano.

[32] **POM**: Parallax Occlusion Mapping genera falso relieve gracias a alteraciones en las UVs. Es más realista que Bump (mapas de normales), pero también más pesado en el rendimiento.
[33] **Displacement**: genera relieve real extruyendo la geometría de un modelo para que concuerde con el de la textura. Máximo realismo, pero muy pesado en rendimiento. Requiere subdivisión para funcionar.

Como escribir código se escapa del enfoque general del artista 3D, que por lo general trabaja con nodos visuales cuando se requieren de acciones técnicas, se han dejado de lado estas ambiciones para el producto. La solución ha sido emplear el Height map para conseguir relieve adicional, aunque obtenga resultados menos realistas y con menos protuberancia percibida. Si se combina con un uso amplificado del mapa de normales, se puede conseguir un efecto lo suficientemente creíble.

Se ha comprobado que valores superiores a 2.5 para el multiplicador del Normal map y por encima de 0.4 para el Height map pueden conllevar errores de iluminación y de visualización, tanto en bakeado como en tiempo real.

Por último, el Height map se comprobó que funciona mejor para relieves fuertes cuando está muy desenfocado. Una textura crispy para esto no consigue resultados creíbles. Al resultar una imagen muy difusa, no es necesario que la resolución sea la misma que la del Albedo o el Normal map: Height puede tener la mitad o menos. Esto ahorra espacio en disco y hace que las texturas carguen más rápido en el buffer del motor.

Con respecto a máscaras, ocurre lo mismo que con POM, es necesario Shader Graph u otro editor de grafos. La solución es añadir las máscaras al paquete e indicar en la documentación unas instrucciones acerca de cómo emplearlas, pero sin adjuntar un shader que lo haga para no tener dependencias en el paquete.

3.5.5.1.3 Otros aspectos técnicos a aplicar a los assets en Unity.

Se descubrió que hay una serie de procesos técnicos obligatorios para publicar assets para videojuegos. Si bien pueden ser repetitivos de aplicar a cada prop y en conjunto llevan mucho tiempo, a veces incluso más que texturizarlo, son necesarios si el cliente ha de usarlos. Estos procesos son: añadir colisiones, aplicar la resolución de lightmap por tamaño de prop, crear prefabs para añadir utilidades y montajes habituales, y crear LODs.

COLISIONES

Para que un objeto dentro de un videojuego no sea atravesado por el jugador y el resto de elementos ha de tener una colisión asociada. Unity, a diferencia de Unreal, no crea colisiones automáticamente ni permite guardar las colisiones en el archivo de importación, por lo que es necesario crear “prefabs” que las contengan por cada uno de los props originales. Un prefab es una plantilla reutilizable de un objeto que guarda su configuración (malla, materiales, colisiones, scripts, etc) para poder instanciarlo fácilmente en la escena o en otros proyectos.

Para añadir una colisión a un objeto de la escena o a un prefab se le añade un componente del tipo “Collider”. “Mesh Collider” calcula automáticamente la forma del modelo y le añade una colisión con su forma exacta, pero consume mayor rendimiento y usarlo en muchos props es muy poco recomendable; es mejor dejarlo para props complejos. En cambio, los colliders simples, como “Box Collider” (un cubo), “Sphere Collider” (una esfera), y “Capsule Collider” (cilindro con bases redondeadas), consumen muchos menos recursos.

La buena práctica es tratar de manualmente recrear la forma del objeto con colliders simples. Si es posible con Box Collider, que es el más sencillo de todos, mejor. Una vez

terminado el proceso, se guarda el objeto entero como prefab en la carpeta de proyecto (ver más adelante). Cuando hay muchos props en el paquete, como en este caso, se trata de un proceso artesanal de varias horas.

PREFABS

Los prefabs son útiles para juntar funcionalidades y para prefabricar combinaciones de objetos que suelen ir juntos en uno solo. Por ejemplo: el modelo de una lámpara junto al de una vela ya colocada en su lugar correspondiente de la lámpara y con un VFX de llama y una luz; o un edificio entero en base a juntar sus partes modulares. De esta manera también se le explica visualmente al cliente cuál es el uso intencionado de ciertos props.

Crear un prefab en Unity es tan sencillo como seleccionar todos los assets que se pretenden juntar en la escena o en la jerarquía de objetos, y arrastrarlos con el ratón a la carpeta del proyecto. Automáticamente se crea el prefab con esos objetos en cuestión y se abre su ventana de edición específica.

Antes de crear el prefab es buena idea anidar todos sus futuros componentes bajo un mismo objeto padre. De esta manera se puede editar después el pivote del prefab con respecto de los objetos que contiene, lo que facilita mucho su colocación posterior en la escena 3D, especialmente si es grande. Para ello se hace clic derecho con todos los objetos que van a formar parte del prefab seleccionados a la vez, y se le da a “Create Empty Parent”. Ojo, esta opción no aparece si uno de los objetos ya es un prefab con anterioridad. En este caso es necesario primero darle clic derecho al prefab en cuestión en la jerarquía de la escena y seleccionar “Unpack Prefab Completely”. Esta opción es muy útil para disponer de sus componentes ya instanciados en la escena de forma separada y sin dependencias (Unreal no lo permite con sus Blueprints, su alternativa a los prefabs).

Una vez creado el prefab, toca colocar su pivote al lugar intencionado (a los pies, en su punto de rotación lógico, etc), que no se puede resetear de manera automática mediante su componente Transform como se haría en la escena 3D. En la ventana de edición de un prefab es necesario seleccionar todos los componentes hijos del pivote y moverlos manualmente hasta que el punto de origen de su objeto padre se halle en el lugar intencionado de pivote para el asset. No se pueden ver al mismo tiempo ambos pivotes ni se permite editar los valores de la posición con cambios numéricos, por lo que la única opción es mover los objetos previamente seleccionados mediante su gizmo[34] de movimiento. Usar la cuadrícula del suelo como referencia ayuda.

[34] **Gizmo**: herramienta visual en programas 3D que permite mover, rotar o escalar objetos en la escena usando ejes o manejadores visibles. Por ejemplo, el gizmo de movimiento muestra flechas para recolocar objetos fácilmente. También hay gizmos que meramente indican la posición de objetos funcionales en la escena, como el caso de luces (gizmo de bombilla), decals (papel doblado), etc.

Proceso resumido de creación de Prefab:

1. Seleccionar todos los componentes deseados en la escena o la jerarquía.
2. Crear un “Empty Parent” que los contenga para tener un pivote común. Si uno de los componentes ya era un prefab, usar “Unpack Completely” primero.
3. Arrastrar el nuevo objeto a la carpeta de proyecto.
4. Mover manualmente los objetos anidados para que coincidan con el pivote del objeto padre. Dejarlos en el punto idóneo para que el prefab tenga su pivote.
5. Añadir la funcionalidad deseada si aplica: colisiones si no hay, luces, scripts, etc.

LODs

Se comprobó que para publicar un asset como “listo para videojuegos” era necesario crear LODs. Los Level of Detail son importantes para reducir la carga de rendimiento, puesto que permiten al motor cambiar entre versiones con diferente escala de detalle de un mismo prop en función de la distancia. Esto permite un cambio progresivo cuanto más se aleje la cámara a versiones con menos carga poligonal, ya que en la lejanía, el detalle extra no se captaría de todas formas.

Todo prop que vaya a visualizarse desde muy lejos (arquitectura) o que tenga una carga poligonal alta (mayor de 1-2k de tris) es recomendable que tenga LODs. También, objetos que tengan una densidad relativamente baja, pero que se van a instanciar muchas veces en la misma escena (cajas, barriles, etc) es bueno que los tengan. El resto de props no es necesario, además, el archivo pesa más si contiene LODs, por lo que si aportan poco es mejor no incluirlos.

Unity permite gestionar LODs mediante el componente LOD Group, que se ha de añadir al prefab del prop. No se puede configurar desde el archivo original de importación como se haría en Unreal. Tampoco permite empaquetar los modelos automáticamente en el mismo archivo. Unity requiere de una malla importada separada por cada LOD del modelo a aplicar. Es por esto que solo se recomienda hacer Level of Detail en Unity para los props que han de llevarlos sí o sí. El tamaño añadido en disco tampoco es muy notable, ya que los modelos ocupan muy poco en comparación con las texturas, pero se trata de un trabajo extra considerable configurarlo así.

No obstante, es posible crear LODs de forma automática y nativa en Unity 6, sin falta de crearlos en el software 3D, pero se aplican desde las propiedades del proyecto, por lo que no son exportables a paquete como tal. Otra alternativa sí exportable es crearlos con la herramienta externa AutoLOD, que una vez aplicado el LOD Group, que sí es nativo a Unity, ya no se generan dependencias de paquete. Entonces se puede eliminar AutoLOD del proyecto sin problemas. Al tratarse de un asset externo, se añaden links formativos sobre su funcionamiento en el Anexo.

El componente LOD Group de Unity es sencillo de usar una vez se tienen hechos los distintos niveles de detalle del modelo y están cargados jerarquizadamente al prefab. Por convenio, la nomenclatura dicta añadir el sufijo “LOD” seguido inmediatamente de su número, partiendo desde 0 como el más detallado y que se verá cuando la cámara esté más próxima, hasta el menos detallado. Ejemplo: *mesh1_LOD0, mesh1_LOD1, mesh1_LOD2.*

Figura 66. *Estructura de LOD en Unity.*

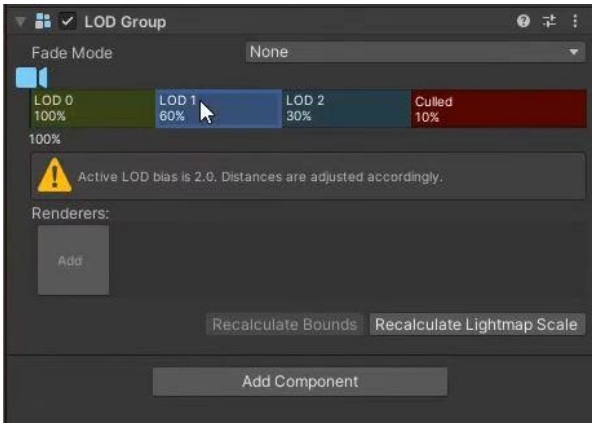


Figura 67. *Componente LOD Group, Unity.*

LIGHTMAPS

En Unity se puede especificar que un objeto es “Static” en el prefab, cuando aplique. Un objeto estático recibe iluminación bakeada. Ya que se han creado prefabs de cada uno de los props para tener las colisiones, no es molestia dejar marcados aquellos que no se van a mover como static. No obstante, no es posible dictaminar del todo si el cliente va a mover o no de forma dinámica un prop en su juego, por lo que no sería necesario marcar prefabs como estáticos, pero sí que es una buena práctica para props de naturaleza decorativa o inamovible. Por ejemplo, marcar paredes y suelos como estáticos muy probablemente no caiga en error. Esta opción Static se encuentra arriba del todo a la derecha en el Inspector, junto al nombre del objeto seleccionado. Atención, no es accesible desde el fbx, solo desde el prefab.

Un prop estático ya aparecerá en los lightmaps. Para especificar la resolución que un prop ha de tener en un lightmap de Unity se ha de ir a las opciones de Mesh Renderer del prefab. Esto aparece en el editor. Se ha de marcar “Contribute Global Illumination” en la pestaña Lighting, y asignar la resolución en la pestaña Lightmapping. A diferencia de en Unreal, no se especifica un número relacionado directamente con la resolución (64, 512, 1024, etc), sino que va en valores del 0 al 4 (se puede poner números mayores, pero no está recomendado).

Tamaño del prop	Valor recomendado Scale In Lightmap	Ejemplos
< 0.3 m (tazas, cubiertos)	0.1 – 0.3	No necesitan detalle, ni sombras finas
0.3 – 1 m (botellas, sacos, sillas pequeñas)	0.5 – 0.8	Suficiente detalle en luz indirecta
1 – 2 m (mesas, barriles, bancos)	1.0 – 1.5	Valor por defecto está bien
2 – 4 m (paredes, vallas, cofres grandes)	1.5 – 2.5	Aumentar si tienen textura o sombras finas
> 4 m (suelos, muros largos, tejados)	3.0 – 5.0+	Para evitar pixelado o artefactos

Tabla 24.Valores recomendados para Scale in Lightmap por tamaño de prop, Unity.

3.5.5.1.4 Estructura de paquete y nomenclatura en Unity

La estructura de un paquete no es obligatoriamente la misma, pero Unity tiene unas directrices recomendadas que repite tanto en la Asset Store Submission Guidelines como en Unity Manual-Asset Packages.

```
Assets/
└── [NombreDelPaquete]/
├── Art/
│   ├── Models/          ← FBX, prefabs vacíos, etc.
│   ├── Textures/        ← Texturas sueltas, PNG, TGA, etc.
│   ├── Materials/       ← Materials de Unity
│   └── Shaders/         ← Shaders personalizados (si hay)
── Prefabs/            ← Prefabs listos para arrastrar
├── Scenes/            ← Escena de ejemplo o test
├── Scripts/           ← Scripts C# (si aplica)
├── Animations/        ← Clips de animación, controllers
├── Audio/             ← Sonidos o música si incluye
├── Documentation/     ← Manual, changelog, guía de uso
└── Editor/            ← Scripts editor personalizados (si hay)
```

Figura 68. *Estructura estándar de paquete recomendada por Unity Asset Store.*

Unity recomienda emplear esta estructura cuando hay reutilización de materiales, pero sugiere empaquetar todo lo referente a un asset en una sola carpeta específica si es que el asset no es reutilizable. Por ejemplo, un modelo con material y textura que solo emplea ese mismo modelo, puede ir contenido en la misma carpeta que ambos; no es necesario separarlos por tipo.

Para el presente proyecto se ha optado por una solución de estructura mixta entre los dos acercamientos. Se han imitado otros assets de la tienda, incluyendo una carpeta de "Shared" para contener los assets que sí son reutilizables, y luego se ha seguido el empaquetado estándar para los assets únicos no reutilizables.

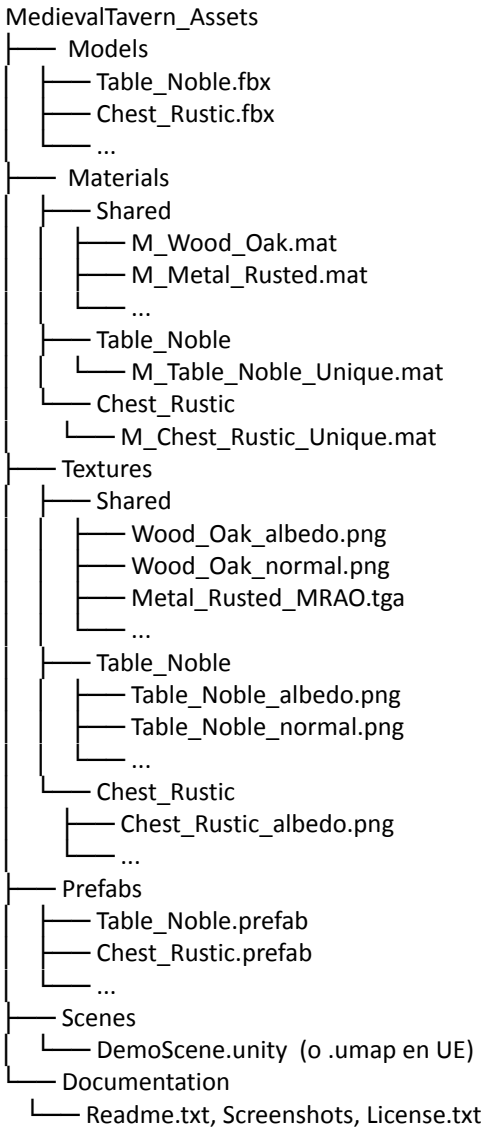


Figura 69. *Estructura de paquete empleada en el proyecto.*

Es importante exportar desde la raíz del paquete y nunca desde más arriba o se podrían exportar assets innecesarios que colisionaran con los del cliente. Además, Unity Asset Store obliga a encapsular todo el paquete en una única carpeta, por lo que exportar desde más abajo no es una opción tampoco; por no hablar de que eso podría conllevar desorden de assets y una exportación incompleta.

Es obligatorio para publicar en las categorías 3D, 2D, VFX, Animation y Templates de la Asset Store incluir una escena de prueba donde el cliente pueda ver con facilidad cuál es el uso ideal de los assets, el intencionado por el autor. De esta manera, el usuario puede partir de combinaciones de prefabs preconcebidas y ahorrar tiempo, además de descubrir usos que no se le habrían ocurrido por su cuenta. También es extremadamente recomendable incluir un "readme", un archivo de texto donde se explique la instalación y las instrucciones del paquete. A diferencia de Fab para Unreal, no es necesario que la escena de prueba contenga expuestos todos los objetos útiles del paquete, solo una mayoría subjetiva razonable.

3.5.5.2 Unreal Engine 5

3.5.5.2.1 Exportación de Blender para Unreal Engine 5

Los modelos exportados desde Blender tienden a dar una advertencia de importación en Unreal por los ya citados en varias ocasiones "Smoothing Groups". Blender utiliza su propio sistema de suavizado de normales y al importar modelos al motor, este no las detecta como correctas y lanza el warning.

Figura 70. *Warning de que no se detectan smoothing groups al importar modelos de Blender en Unreal.*

Se trata de un problema menor, ya que el FBX exportado desde Blender sí que contiene información de normales y suavizado, solo que de otra forma diferente a la que Unreal espera. Se ha comprobado que la traducción automática de normales que hace el motor al recibir estos modelos de Blender es correcta en la vasta mayoría de los casos.

Exportar en modo "Face" y marcar la opción "Tangent Space" (no funciona si hay NGons) en "Geometry" en lugar de exportar en modo "Normals Only" (para Unity) tiende a resolver la aparición de los warnings, casi siempre. Una vez importado, el FBX se cambia automáticamente a formato uasset, que al ya estar procesado por Unreal no genera warnings cuando el cliente lo vaya a importar.

Como exportar dos veces cada archivo con modos diferentes es una carga de trabajo adicional, se optó por importar en Unreal los assets tal y como se habían exportado para Unity. Salvo cinco excepciones, no hubo ningún problema.

Para poner remedio a las excepciones que se sombreaban mal con modo “Normals Only”, se procedió a hacerlo en modo “Face”. Esto arregló la mayoría de los casos.

Otro problema habitual son cortes de UVs marcados de forma brusca en superficies redondeadas. Esto se debe a que el smooth simple de Blender no se traduce bien a MikkTSpace, que es el formato que espera Unreal. Es por esto que para estos casos concretos, se recomienda usar siempre Auto-Smooth. Si se sombrean aristas duras en la superficie redondeada, simplemente se ha de subir el ángulo de aplicación del suavizado en la ventana de normales (menú inferior derecho, icono de triángulo verde invertido). Esto consigue un visualizado similar al de smooth simple, pero sin errores en Unreal.

En un único caso fue necesario aplicar un modificador “weighted normals” de Blender para reforzar las normales. Combinado con “Add custom normal data” arregló el problema de visualización.

Los ejes aplicados de exportación en el FBX que tan críticos habían sido para Unity, no constituyeron un problema en Unreal, los tradujo todos bien.

El flujo utilizado resumido ha sido el siguiente:

1. Utilizar los FBX exportados para Unity.
2. En caso de mala visualización de normales tras importar, volver a Blender y exportar las normales en modo Face en lugar de en Normals Only.
3. En caso de necesitarse, añadir y aplicar un modificador weighted normal desde Blender. Añadir custom normal data.
4. Evitar Smooth simple en props redondeados o aparecen cortes.

3.5.5.2.2 Materiales en Unreal Engine y el método MRAO

El empaquetado estándar, aunque no obligado, que se emplea para ahorrar espacio en disco con las texturas en Unreal Engine es el formato MRAOH. La H no es necesaria si no se va a emplear el mapa de alturas.

Canal	Mapa Empaquetado
R (rojo)	Metallic
G (verde)	Roughness
B (azul)	Ambient Occlusion
A (alfa)	Height Map

Tabla 25. *Empaquetado de mapas en Unreal con formato MRAOH.*

Como Roughness tiende a recoger mayor detalle que el resto de mapas que se suelen empaquetar, comparable a veces a la información que podría contener un Albedo o un Normal, es el que se asocia al canal verde. Este canal resulta que tiene mayor precisión (6 bits en lugar de 5) en los formatos de compresión que usa Unreal Engine (DXT1, DXT5, BC7, etc) porque el ojo humano es más sensible al verde que al rojo o al azul, y por tanto, se prefiere almacenar en él mayor detalle.

Para montar las texturas en estándar MRAOH en Photoshop, hay que seguir el mismo proceso que en el apartado 3.5.5.1.2, pero con la nueva distribución de mapas.

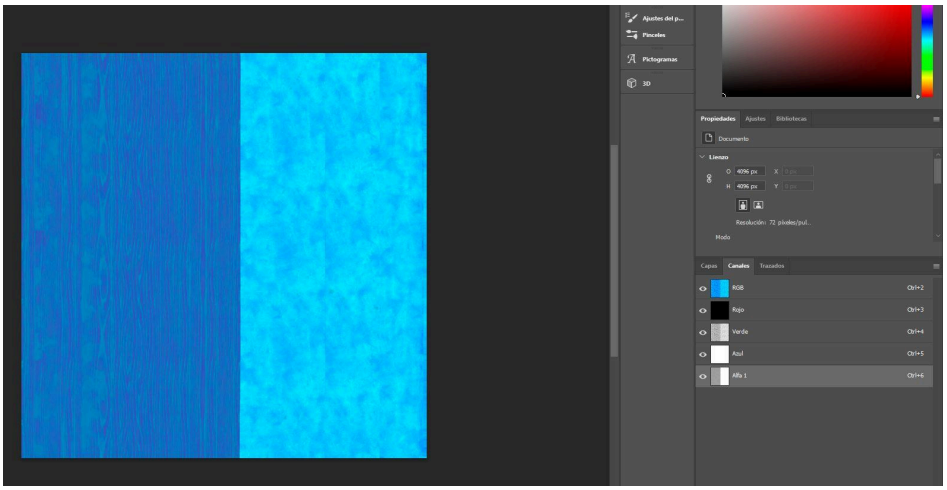


Figura 71. *Ejemplo de distribución de mapas en MRAOH en Photoshop. Trimsheet principal para los edificios modulares.*

En el ejemplo anterior, Fig., se ha rellenado de negro el canal rojo (metallic) porque se trata de un trimsheet de yeso y madera, que son materiales no metálicos. Al no incluirse un mapa Ambient Occlusion se ha dejado en blanco puro el canal azul para que no se oscurezca el material al multiplicarse el mapa con el resto de texturas. Los canales verde y alfa llevan el Roughness y el Height, respectivamente.

Una vez se importan las texturas en Unreal, es necesario cambiar el modo de compresión de los mapas que no sean BaseColor al modo “Masks” (ver Fig.). Esto es similar a indicar Non Color en Blender. De esta manera no se usa sRGB. Para hacerlo, se le da doble clic en la textura desde la carpeta del proyecto. Este menú que se abre, también sirve para ver si la imagen se ha empaquetado correctamente y no falta el canal alfa (arriba a la derecha). Si la “A” no aparece, la textura ha sido mal exportada. Por otro lado, el mapa de normales se configura automáticamente al importar, no hay que hacerle nada, o si no se detecta entonces, se configura al asignarlo a un material. Aparece un prompt y se le ha de decir que sí. Los Base Color pueden permanecer en la compresión por defecto, que es DXT1. Si un mapa MRAOH no se cambia a Masks, se visualizará mal.

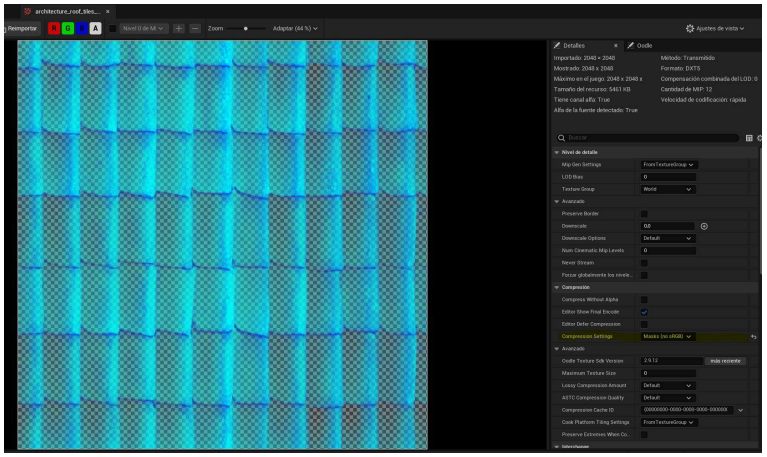


Figura 72. *Opciones correctas de importación de una textura MRAOH en Unreal Engine.*

No obstante, por defecto Unreal Engine no interpreta las texturas en formato MRAOH y es necesario colocar los nodos a mano en el grafo del material. No es tan sencillo como arrastrar las imágenes a slots preparados como se hizo en Unity.

Si el material MRAOH lleva mapa Height, que va en el canal alfa, es muy recomendable exportar en formato TGA para mantener la información en ese canal sin cambios, pero si no cuenta con ese mapa, se puede exportar directamente en PNG para que ocupe menos.

En Unreal es posible modificar los Shaders de forma visual por nodos sin requerir de dependencias externas, por lo que sí se han hecho máscaras de color y parámetros inteligentes para variar la fuerza de cada mapa.

Lo primero es definir un Shader en estándar MRAOH (ver Fig.). Se ha de crear un “material” a secas, con clic derecho en la carpeta de proyecto, un material master que establecerá las dependencias y funcionamiento de las texturas y del que heredarán el resto, que serán “instancias” de material. Una vez creado, se abrirá su menú de edición haciendo doble clic y se empezará a añadir nodos (clic derecho). Primero se crearán tres del tipo Texture 2D, uno para el BaseColor, otro para asociar el mapa empaquetado, y otro para el mapa de normales. Después, se conectan las salidas RGB del BaseColor y el Normal a sus respectivas entradas en el Shader, pero en el caso del MRAOH, se conecta cada canal de color a la entrada que represente según el estándar de empaquetado: rojo-> Metallic, verde-> Roughness, azul-> AO. Alfa no se conectará a nada por el momento. La salida RGB se no se conecta. Es necesario

asignar texturas al material master que coincidan con el tipo de mapa específico aunque sean solo de placeholders, o dará error de compilación el shader. Es decir, se necesita conectar un mapa real de normales al Param2D que va conectado a la entrada de normales del shader y viceversa. Se pueden añadir Scalar Parameters para controlar la fuerza de los mapas de texturizado mediante una multiplicación numérica. Ajustar la intensidad del mapa de normales, en cambio, requiere de un Lerp entre la textura y el vector (0.5,0.5,1), con el Scalar Parameter como entrada de Alpha.

Aunque Unreal Engine sí permite unos resultados mucho más ambiciosos sin dependencias externas de paquete, se optó por que la versión de Unreal no fuera radicalmente diferente de la de Unity. Se trató de mantener cierta coherencia. Esto dejó fuera cosas como un Parallax Occlusion Mapping (POM complejo), Displacement, o shaders de cristales (en Unity no se puede sin Shader Graph o recurrir a código). Esta última limitación provocó que se optará por cerrar las ventanas de los props de arquitectura con contraventanas y rendijas de madera, aunque curiosamente es más fiel históricamente de esta forma. Los cristales no aparecieron en los edificios comunes hasta bien entrado el S.XVII, y no fueron un estándar hasta el S.XIX. Antes, era cosa solo de catedrales y edificios de la más alta nobleza.

Entonces, en Unreal se optó por replicar el método de normales combinadas con Height para falsear relieve, que es lo mismo que se pudo hacer en Unity. No obstante, se adjuntan recursos formativos para relieves extremos y realistas por POM en el Anexo.

Hacer una máscara en Unreal requiere de usar un nodo Subtract que reste la textura de la máscara de la textura original. Luego, por separado, un nodo Color se multiplica con la máscara mediante un nodo Multiply para decirle a qué tonalidad cambiar. Los resultados de la sustracción y la multiplicación se combinan en un nodo Linear Interpolate. Este resultado se puede encadenar con más máscaras si se quiere, antes de asignarlo al BaseColor del Shader.

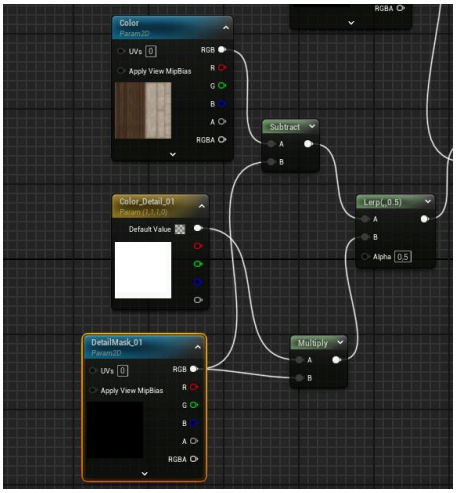


Figura 73. *Máscaras en Unreal.*

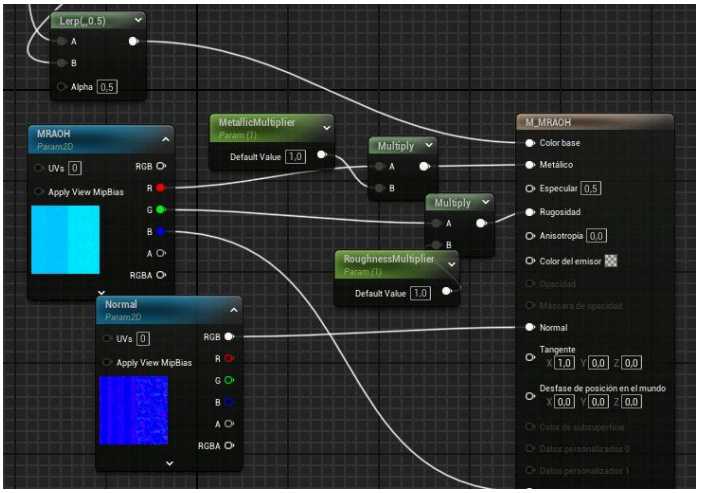


Figura 74. *Setup general de MRAOH en Unreal.*

Emplear el mapa Height como se hizo en Unity, aquí requiere de sacar la textura MRAOH duplicada en un nodo del tipo Texture Object Parameter y emplear su mapa alfa

(Constant 4 Vector en 0,0,0,1) para hacer una mezcla con el mapa de normales gracias a los nodos Normal from Heightmap y Blended Angle Corrected Normals.

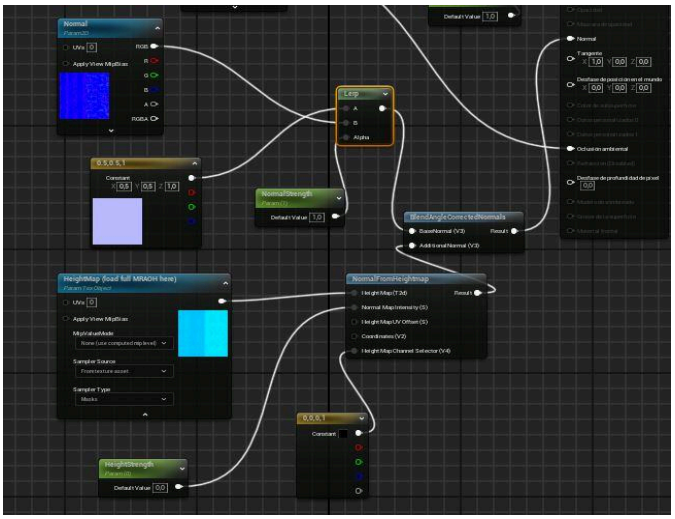
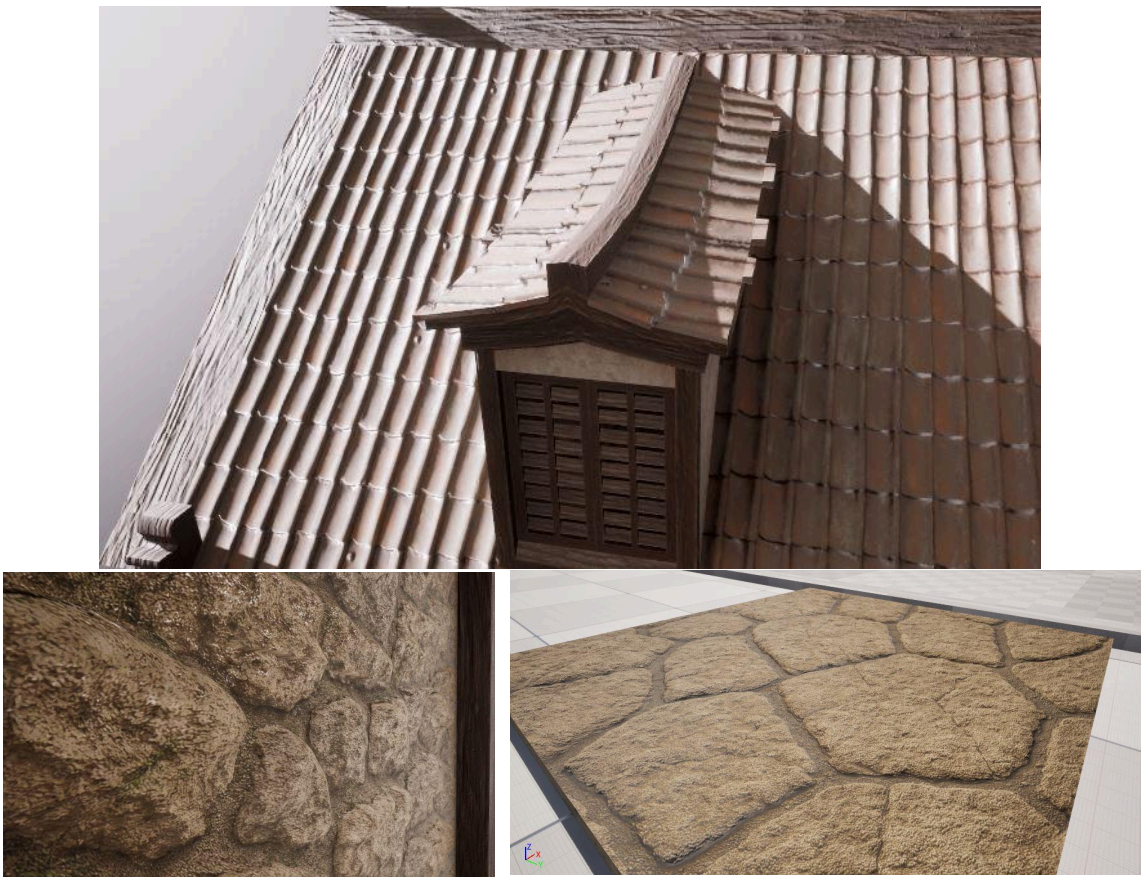


Figura 75. *Setup de Normal combinado con Heightmap en MRAOH, Unreal.*

Luego, para hacer transparentes los Card Decals, fue necesario crear un shader maestro auxiliar. Para visualizar la transparencia, hay que ponerlo en modo Translucent en la pestaña de Detalles; por defecto viene en Opaque. Además, el canal alfa de la textura cargada como BaseColor ha de ir conectado al Opacity del shader. Por supuesto, la salida RGB va al Color Base del shader. Para evitar brillos raros, es preferible poner el shader en modo Unlit, aunque en algunos casos concretos se puede querer que reaccione a la luz y se deja en Lit.

Se pudo introducir una versión sencilla de Parallax Occlusion Mapping (POM) en algunos materiales clave sin desprestigiar mucho a la versión de Unity. Como se trata de una explicación compleja sin apoyo visual, se ha grabado cómo conseguir el efecto en un vídeo consultable en el Anexo. No obstante, la idea general es la siguiente:

- Crear un nodo Parallax Occlusion Mapping.
- Asignarle la textura MRAOH como Texture Sample, no como Param2D.
- Agregar un nodo Constant4Vector en (0,0,0,1) conectado a la entrada HeightMap Channel para decirle a Unreal que el mapa de alturas está guardado en el canal alfa.
- Crear Scalar Parameters para controlar las entradas MinSteps, MaxStep, ShadowSteps y Shadow Penumbra.
- Asignar por defecto los valores 32 y 64 a MinSteps y a MaxSteps, respectivamente. Los otros dos parámetros no son importantes para un material con POM simple.
- Crear nodos NameReroute y asignarlos a cada una de las entradas del lateral derecho del nodo POM. Se les da el nombre de cada entrada.
- Añadir un nodo ParallaxUV, que es uno de los NameReroutes que se crearon en el paso anterior, y se conecta a la entrada UVs de cada una de las texturas del Shader, es decir, a BaseColor, MRAOH, Normal, etc. Es indispensable hacer esto para que funcione.



Figuras 76-78. *Ejemplos de POM en la versión de Unreal del producto.*

3.5.5.2.3 Otros aspectos técnicos a aplicar a los assets en Unreal.

Como ya se ha comentado, existen ciertos aspectos específicos a motores que se han de añadir a cada prop para poder anunciarlo como optimizado para videojuegos. Estos son la creación de colisiones, la inclusión de LODs para objetos complejos, crear Blueprints para juntar funcionalidades, y aplicar las resoluciones de lightmap a cada prop en función de su tamaño.

COLISIONES

Unreal Engine genera colisiones automáticas en los props por defecto, y además de forma optimizada, que es muy de agradecer. Unity puede crearlas automáticas, pero requiere de Mesh Collider, que consume mucho, y necesita ir en un prefab. Unreal lo guarda en el archivo uasset, que es a lo que traduce los fbx importados. Entonces, no es necesario sacar cada prop a Blueprint (similar a prefab, pero en UE) solo para tener colisiones. En resumen, no es necesario perder tiempo creándolas a mano en la gran mayoría de los casos.

Eso sí, las colisiones simplificadas que hace Unreal Engine 5 no respetan cavidades, sino que tienden a simplificar las formas a cubos, por lo que si un objeto ha de ser transitable

por un agujero, por ejemplo, pues ha de tener una colisión personalizada. Se pueden generar colisiones automáticas complejas, similares a los Mesh Colliders de Unity, desde el menú de edición del asset. Arriba a la izquierda hay un desplegable de colisiones (Collision), donde en la versión 5.5 se ha de dar a "Remove Collision", y después a "Auto Convex Collision". No obstante, esta versión está poco optimizada. Para casos en los que no sea absolutamente necesario, se recomienda borrar la colisión automática como ya se ha explicado, y después optar por añadir formas primitivas desde ese mismo menú, por ejemplo, "Add Box Simplified Collision". Se pueden escalar y recolocar, también se pueden añadir tantas como se quieran hasta tener el objeto completo representado con las formas primitivas.

BLUEPRINTS

El funcionamiento de los Blueprints, en lo que atañe a un artista 3D que solo junta props en su distribución intencionada, es casi idéntico al de los prefabs de Unity. Sí, hay muchas diferencias a nivel técnico y en otros ámbitos, pero para crear un paquete de modelos 3D, no hay tantas.

El procedimiento a seguir es el mismo que el del prefab (ver sección 3.5.5.1.4): anidar los objetos en empty que los contenga para poder modificar luego el pivote, crear el blueprint, ajustar manualmente el pivote de los objetos hijos, y añadir toda funcionalidad que se quiera.

La peculiaridad aquí es que no hay que desempaquetar un objeto que ya fuera un blueprint previamente, de hecho no lo permite, y que la forma de crearlo es ligeramente diferente. En Unreal, con los objetos seleccionados desde la jerarquía o la escena, se va a la barra de herramientas superior general, y se abre el desplegable de Blueprints, que es un icono de tres cuadrados unidos entre sí por líneas. Está junto al desplegable para añadir objetos nuevos a la escena, que tiene un icono de un cubo seguido de un signo "+". Desde ahí, se selecciona la opción "Convert Selection to Blueprint Class". Esto abre un menú de contexto donde se especifica la clase de la que hereda el nuevo Blueprint, que para lo que atañe a conjuntos de modelos se puede dejar por defecto (heredando de "Actor"), y luego, se ha de especificar la ruta de destino del archivo.

Una vez hecho esto, igual que en Unity, se ha de mover manualmente los objetos hijos para conseguir que los modelos estén sobre el pivote del objeto padre en el lugar intencionado para su nuevo pivote. Atención, que no ha de coincidir con el "root" del Blueprint, sino con el objeto que se ha creado que los contiene, que ha de ir anidado bajo el root. De no haberse hecho así, siempre se pueden emparentar los objetos dentro del Blueprint añadiendo un componente del tipo "Scene component" desde el desplegable "Add" (encima de la jerarquía). Este nuevo componente sí tendrá un pivote visible, no como el root.

LIGHTMAP RESOLUTION

Dejar la iluminación prefabricada para ahorrar rendimiento en un juego es fundamental, calcularla en tiempo real es más costoso. A pesar de esto, Unreal cuenta con el sistema Lumen para "real time illumination", que hace bastante bien el trabajo sin comprometer tanto el rendimiento. No obstante, como no se sabe el uso que se le va a dar al paquete de assets, es indispensable prepararlo para que permita iluminación bakeada.

Para ello, lo primero es tener las UVs sin que ninguna isla se solape y manteniendo cierta separación (padding). Ya se explicó en el apartado 3.2.12 cómo añadir un canal auxiliar de UVs sin overlap con paddings recomendados por resolución.

Después, hace falta indicarle al Unreal Engine qué canal de UVs ha de usar en cada uno de los props. Por defecto, el primero es para texturizado y el segundo para lightmaps, pero por experiencia propia, muchas veces aparecen cambiados de orden o incluso en slots vacíos que no existen, por lo que los lightmaps saldrán quemados si se detecta overlap.

También es importante especificar qué resolución tiene cada prop en los lightmaps. Esto, y qué canal se ha de usar se configura desde el menú de contexto del Static Mesh (el fbx importado). Se accede a él haciendo doble clic en la carpeta de proyecto en el prop en cuestión, y mediante la lupa de búsqueda se escribe "lightmap". Aquí aparecerán Lightmap Resolution, que es donde se ha de especificar la resolución del prop, Generate Lightmap UVs, que ha de desactivarse o se crearán UVs que no son las que se hicieron en Blender, y Lightmap Coordinate Index, que es el canal de UVs que se usa para el bake.

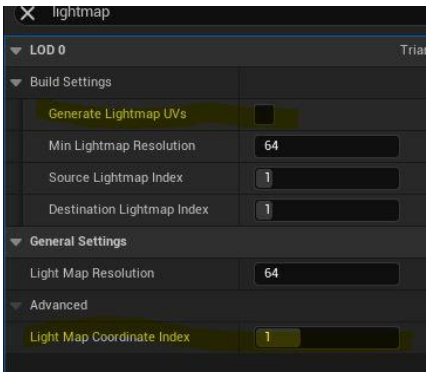


Figura 79 (Izquierda). *Lightmaps quemados en la escena de demostración del proyecto tras bakear con UVs incorrectas.*

Figura 80 (Derecha). *Opciones para lightmaps en Unreal.*

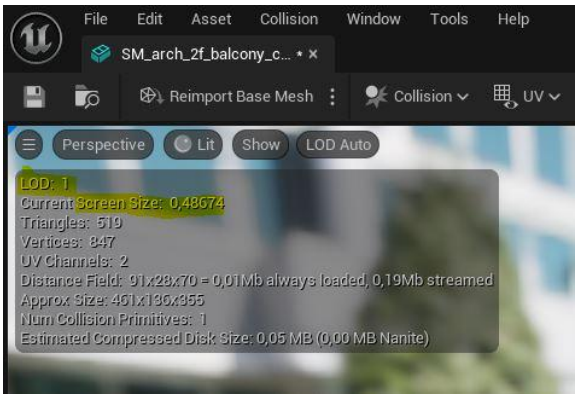
Cuando no se seleccionan bien las UVs, los props quedan quemados al generar Lightmaps en la escena. Esto quiere decir que van a tener una película blanca multiplicada sobre su superficie allá donde incidiría la luz.

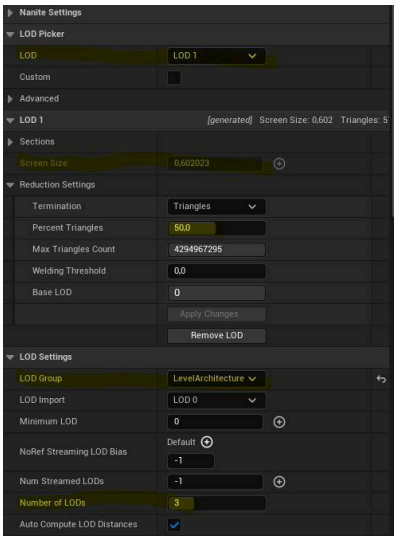
No obstante, la generación de los lightmaps en sí corre a cargo del cliente. Solo hay que preparar los props para que lo permitan.

LODs

Se pueden crear LODs manualmente en Blender mediante el modificador “Decimate”, que respeta UVs y texturas, pero Unreal los crea a la perfección en la mayoría de los casos de forma automática. Eso sí, es necesario ajustarlo, puesto que muchas veces la distancia de cambio de un nivel de detalle al siguiente puede optimizarse bastante. La cantidad de reducción, también.

En el menú de contexto del Static Mesh, se va a la sección de LODs y se selecciona el LOD Group que mejor se ajuste al tipo de prop en cuestión. Por defecto viene en None, pero al cambiarse la opción a uno de los grupos predefinidos pasa a tener LODs generados automáticamente. Para modificar manualmente la distancia a la que se produce el cambio de un LOD a otro, es necesario cambiar la opción LOD Auto en la sección LOD Picker a aquel que se quiera editar. Ahora, aparece una opción Screen Size que hace referencia al tamaño que tiene que ocupar un prop en pantalla para que cambie al siguiente nivel de detalle. Este número muchas veces se puede aumentar bastante sin que el cambio resulte perceptible a simple vista. También se puede modificar la cantidad de reducción. Para afinar la distancia, se pone la opción en Auto de nuevo y se pone y se quita zoom en el visualizador 3D. Justo arriba a la izquierda aparece el número de triángulos que se están visualizando en cada momento y el LOD en el que está (ver Fig.73).





Figuras 81, 82. *Opciones de LODs en Unreal Engine.*

3.5.5.2.4 Estructura de paquete y nomenclatura en Unreal Engine

Unreal Engine tiene ciertas peculiaridades acerca de la nomenclatura de los archivos, y es que obliga por convenio a añadir prefijos a los archivos para ordenarlos por tipo: “SM” (Static Mesh) para mallas de modelos, “T” para texturas, “M” para materiales maestros, y “MI” para instancias de material (hay muchos más, pero no atañen). Además, el mapa Albedo o Diffuse del estándar PBR aquí se denomina obligatoriamente “BaseColor”, el resto se denominan igual. Los mapas empaquetados se denominan MRA si no lleva Height y MRAOH si lo lleva, pero son texturas opcionales: se pueden emplear los mapas estándar del PBR por separado como Roughness, AO o Metallic, pero no empaquetarlos es menos eficiente.

Ejemplo: “*T_VarnishedWood_BaseColor.png*”

En cuanto a estructura de paquete, en Unreal tampoco se ha comprobado que haya un estándar rígido. Se han revisado varios ejemplos profesionales y sí que tienen un estructuración lógica y común en ciertos aspectos, como la separación de mapas de juego (niveles) de los assets, o la separación en carpetas de mallas de modelo, texturas y materiales, pero no parece haber un convenio férreo establecido.

De cara a este proyecto, que tiene assets compartidos, pero también modelos texturizados de manera única y no compartida, se ha optado por un acercamiento mixto en la estructura, igual que se hizo para Unity. Una jerarquía de archivos mixta y en la línea de la de los paquetes examinados para investigación es:

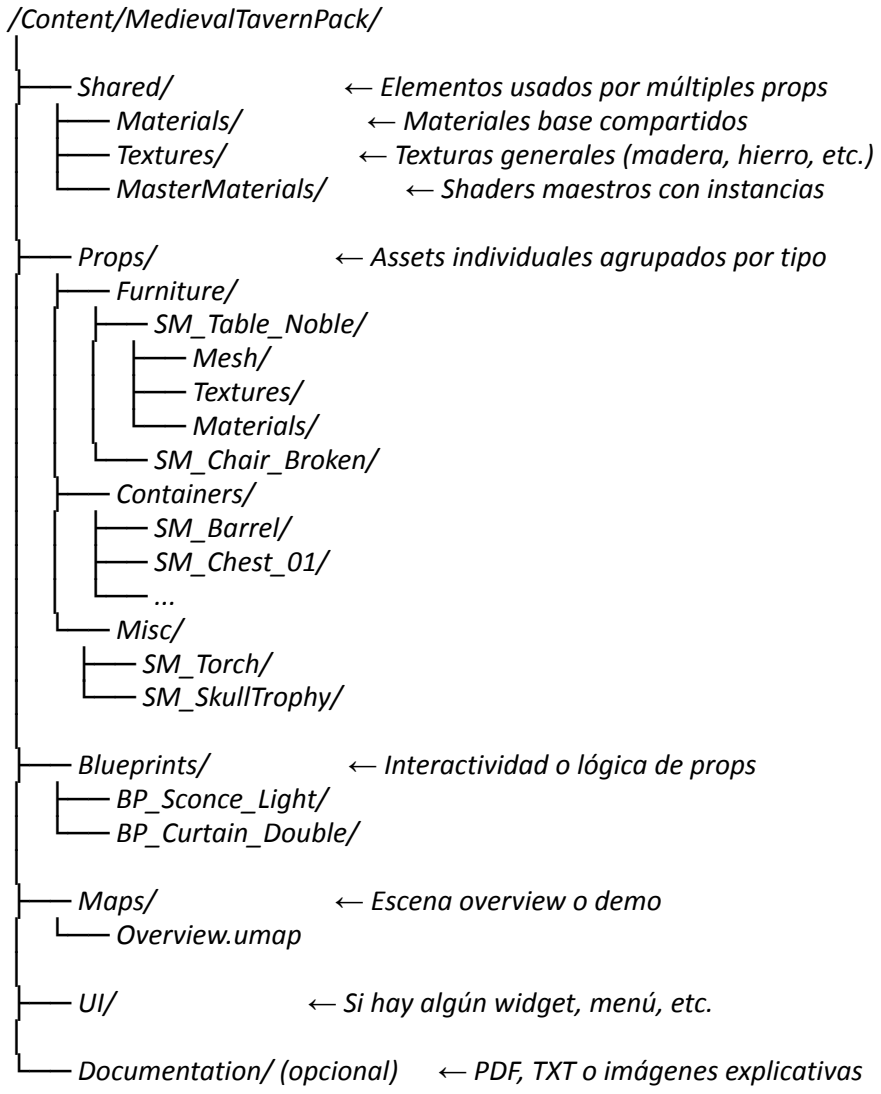


Figura 83. *Jerarquía empleada para el paquete en Unreal Engine 5.*

Como se ve en el siguiente apartado, es necesario incluir un nivel Overview donde se muestren todos y cada uno de los componentes que ofrece el paquete. Además, es muy recomendable, pero opcional, añadir un nivel Demo en el que se enseñe el uso intencionado de los assets, es decir, una escena de prueba montada tal y como es la intención del autor y con iluminación pre-renderizada para que luzcan lo mejor posible los assets. Los resultados visuales pueden verse tanto en el vídeo del Anexo como en la sección 3.5 Resultados. No se explica cómo se ha montado la iluminación y la escena por motivos de alcance del documento, ya que esto se mete de pie en la parte de motor, pero se adjuntan recursos formativos en el Anexo.

3.5.6 Venta y Distribución en Unity Asset Store y Fab

Muchas de las exigencias de Fab y de Unity Asset Store, los portales web donde se va a vender el asset, comparten consideraciones generales comunes. Tras una investigación se ha concluido que son:

1. No hay ningún contenido con licencia que requiera atribución. Esto quiere decir que no puede haber material de creación ajena aunque sea Creative Commons (CC). Deberá ser Creative Commons tipo 0 (dominio público y sin atribución). CC BY o CC BY-SA son gratuitas pero no servirían. Hace falta corroborar que los archivos CC0 realmente lo son, algunas webs engañan y puede haber problemas legales.
2. Todo contenido ajeno, que ha de ser CC0, es mejor referenciarlo en la documentación por motivos de transparencia, pero no obligatorio.
3. Contenido hecho por IA referenciado como tal. Además, ha de contener valor humano.
4. Contenido completo y sin errores visibles.
5. Alcanzar un estándar mínimo de calidad que es subjetivo en base a revisión manual por parte del portal de venta.
6. Modelos con unidades coherentes y transformaciones aplicadas.
7. Texturas en un estándar PBR.
8. Salvo en casos muy concretos, se prohíbe el uso de texturas en formato comprimido.

Algunas obligaciones para con el producto son específicas por motor.

En Unity son:

1. El asset deberá estar publicado para una versión mínima que ha de comprobarse. A fecha de junio de 2025, ha de ser posterior a la 2021.3. En este caso se usa la 6000.1.4f1 del 21 de mayo de 2025
2. No se admiten errores o advertencias en consola tras importar el paquete.
3. Ha de haber una escena de prueba.

4. No puede superar las 6gb; a fecha de junio de 2025. Comprobar siempre en el momento de publicación. No es posible exceder este límite ni bajo revisión manual. Entonces, la solución es dividir el paquete en varios productos en lugar de en uno solo.
5. El contenido creado con IA deberá quedar reflejado en el campo “AI Description” y llevar trabajo humano de valor detrás. No sirve con subirlo tal cual se genera.
6. Se ha de incluir en la documentación del paquete un fichero de texto “Third-Party Notices.txt” donde se indiquen todos los assets que sean de creación ajena, que deben ser CC0.
7. Los modelos deberán estar en uno de los siguientes formatos sí o sí: fbx, obj, dae, abc. Unity soporta más extensiones, pero no se permiten en la Asset Store.
8. Transformaciones aplicadas y a escala 1. Las unidades han de ser coherentes (1=1m).
9. Pivotes en la zona inferior y centrados, o si es una pieza modular, en su punto lógico.
10. Eje Z positivo hacia delante.
11. Las texturas deberán ir en un estándar PBR. Hay varios, no tiene porqué ser el Metallic-Smoothness de Unity, se puede configurar en shaders manualmente cualquier otro tipo que se considere un estándar PBR.
12. Las texturas no pueden estar en un formato comprimido. Esto deja fuera JPG. Solo se admiten PNG, TGA, PSD y PSB. Esto deja TIFF fuera aunque sea también sin pérdidas.
13. Las texturas han de estar en potencia de 2. Esto deja fuera cualquier textura personalizada para aprovechar memoria. Solo se admiten entonces del orden 2 elevado a “n”. Estas resoluciones son: 1x1, 2x2, 4x4, 16x16, 32x32, 48x48, 64x64, 128x128, 512x512, 1K, 2K, 4K, 8K. Superior a 8K ya no lo admite el propio motor bajo circunstancias normales, pero no se prohíbe el uso de 16K o más. Eso sí, hay que recordar el límite de 6gb.
14. El proyecto ha de estar en una carpeta única, no sirve suelto.
15. No puede haber archivos innecesarios o duplicados.
16. Los assets deben ir separados en carpetas por tipo, pero si uno de ellos va texturizado de manera única, entonces se pueden agrupar el modelo, la textura y el material en una subcarpeta.
17. Se debe añadir una descripción clara del paquete en la web.
18. Debe haber capturas de calidad.
19. Debe llevar etiquetas y estar colocado en categorías relevantes para el proyecto.

La tienda de Epic Games, Fab, que es donde se distribuyen los assets para Unreal Engine, tiene muchísimas directrices más; entre ellas contenido mínimo por tipo de asset. Esto se debe a que es un portal de venta no solo orientado a videojuegos, también distribuye modelos para render y otros ámbitos, incluso paquetes para Unity (para estos tiene las restricciones idénticas a las de Asset Store). La vasta cantidad de información acerca de lo permitido hace inviable referir toda aquí, por lo que se recomienda acceder personalmente desde el enlace del anexo, pero sí que resume lo importante para un modelador aquí debajo.

Todo lo que no se exponga aquí pero sí que se haya mencionado para Unity Asset Store, quiere decir que para FAB también aplica y se ha de tener en cuenta. Las diferencias reseñables específicas de FAB para un modelador y texturizador son:

1. Tamaño máximo de 15 gb, pero se pueden subir tamaños mayores sujetos a una revisión manual para corroborar que realmente hace falta. Si los assets no están debidamente optimizados, no se permiten archivos mayores de 15 gb. Si el paquete no está en formato de Unreal Engine, es decir, se trata de un asset para Unity o de modelos sueltos o para render, el tamaño máximo son 6gb.
2. En función del tipo de asset que se publica y la categoría en la que encajan sus partes hay un contenido mínimo exigible en número de modelos. Por ejemplo, no es lo mismo subir un vehículo que un árbol lowpoly. Cada tipo de asset requiere de un número mínimo de props. Consultar en el anexo para más detalles. Como anécdota, un personaje en primera persona debe contar con al menos 5 variantes de brazos.
3. Se soportan más archivos de 3D y de texturas. Aceptan imágenes comprimidas como las JPG, pero solo para otros usos, no para materiales. Para ver específicamente cuáles, ir a la web.
4. Los paquetes destinados a ser usados en Unreal Engine deben tener una estructura muy específica y obligatoria, con nomenclatura concreta y dependencias (ver documentación web).
5. No es obligatorio subir escenas demo, pero siempre debe haber una denominada “Overview” que exponga todos y cada uno de los componentes del paquete por separado.

Tras comprobar que el asset cumplía con todas las normas antes referidas, se procedió a crear la cuenta de vendedor en ambas webs. Se pide identificación personal y fiscal. Tanto Unity como Epic Games se encargan de gestionar el IVA en base a la localización del comprador. Ellos son los vendedores legales frente al cliente final. El pago recibido es , por tanto, neto. La responsabilidad de declarar la ganancia como autónomo o empresa a Hacienda Española sí que cae a cuenta del artista.

Aspecto	Fab (Epic Games)	Unity Asset Store
Frecuencia de pago	Mensual, 30 días después de fin de mes. Es decir, el pago por lo vendido en enero se recibe al empezar marzo.	Mensual, alrededor del día 15 a mes vencido, es decir, el pago por enero viene el 15 de febrero.
Umbral mínimo	100 USD	100 USD
¿Se pierde el dinero si no se llega al umbral de pago?	Sí, si no se alcanzan los 100 USD en un año (según su contrato, pueden retenerlo).	No, se acumula hasta que se alcance el umbral.
Comisión	12% para Epic (88% para el artista).	30% para Unity (70% para el artista).
Moneda de pago	Dólares (USD).	Dólares (USD).
Método de pago	PayPal, transferencia bancaria.	PayPal, transferencia bancaria.

Tabla 26. *Comparativa de planes de negocio entre Fab y Unity.*

En resumen, Fab es más jugoso por su comisión sustancialmente más pequeña, pero tardan bastante más en pagar y se quedan con los ingresos de las ventas si termina el año fiscal y no se han alcanzado los 100 USD. Unity Asset Store es más seguro para paquetes pequeños.

Capítulo 4. CONCLUSIONES

4.1 Conclusiones del trabajo

El proyecto ha dado lugar a un asset funcional, optimizado y apto para su uso en entornos de videojuegos. Su nivel de calidad permite su integración en proyectos comerciales, cumpliendo los requisitos fundamentales de eficiencia, estructura y presentación visual.

El producto cumple con todas las especificaciones necesarias para considerarse “Game Ready”, es decir, está preparado para su uso directo en juegos. Esto quiere decir que tiene mallas y texturas optimizadas (poca carga poligonal, reutilización de texturas), texel density coherente, hay al menos un canal de UVs limpio (sin overlap) por asset con padding correcto para bakeo, los assets tienen una correcta resolución de lightmap por tamaño, los materiales están configurados en el motor, los props complejos tienen LODs y colisiones personalizadas, las escalas, ejes y posiciones están bien configurados, y la nomenclatura y organización son correctas.

Si bien los assets cumplen a un nivel suficiente con todos los estándares, es posible que en un entorno de alta exigencia de producción real sea necesario retocar algunos aspectos puntuales para adaptarlos a necesidades específicas de optimización o integración. El producto, en alguna de sus partes podría debatirse que no alcanza calidad AAA, aunque sobradamente llega a AA. Un aspecto mejorable de cara a la impecabilidad técnica es la variación de texel density, que en casos de props de muy diferente tamaño y que comparten texturas en el producto puede haber variaciones mayores del 10% por motivos de modularidad. Luego, está el padding, que aunque se ha contemplado, se ha hecho automáticamente en base a valores numéricos, lo que podría no ser perfecto para todas las situaciones de bakeo de lightmaps complejos.

El desarrollo ha implicado una curva de aprendizaje notable, que ha permitido incorporar técnicas y flujos de trabajo no contemplados inicialmente. A pesar de ello, los objetivos fundamentales se han alcanzado con éxito, y el producto resultante constituye una base sólida para futuras producciones con mayor grado de refinamiento.

A continuación se listan en más detalle conclusiones técnicas.

Número aceptable de assets en el paquete.	SÍ, de hecho se exceden. La taberna no se planteó como modular inicialmente.
Está optimizado para juegos.	Sí
Puede anunciarse como "Game Ready"	Sí, con matices.
El peso del paquete excede lo esperado.	No, pero está cerca.
Funciona en Unity.	SÍ
Funciona en Unreal Engine.	SÍ
Es apto para publicar.	Sí
Es apto para vender.	SÍ
Se ha publicado.	Aún no, requiere revisión manual que se alarga en el tiempo. Puede llegar al mes.
El tiempo de desarrollo excede al 150% del esperado.	Sí
El producto es de calidad AAA.	En algunos aspectos sí, en otros no.
El producto alcanza calidad AA.	Sí
La geometría por modelo excede la esperada por tipo de asset con respecto a lo habitual en juegos modernos publicados.	La mayoría no la excede, pero para los que sí lo hacen, que no es por mucho, se han creado LODs manualmente.
Los materiales alcanzan realismo suficiente.	Sí
Hay colisiones personalizadas.	Sí, aunque solo en props complejos.
Hay pivotes lógicos para facilitar el montaje modular.	Sí
Se incluyen prefabs y blueprints.	Sí
Se incluyen LODs personalizados.	Sí, aunque solo en props complejos, para el resto se generaron automáticamente.
Se incluyen demos y escenas de prueba para cada motor objetivo.	Sí

Tabla 27. *Conclusiones generales del proyecto.*

4.2 Conclusiones personales

El desarrollo de este proyecto ha resultado ser especialmente enriquecedor a nivel formativo y personal, ya que ha permitido cubrir la mayoría de carencias previas en cuanto a la creación de un asset optimizado para videojuegos desde cero. Tras su finalización, el autor cree haber alcanzado una base técnica sólida que le permitiría abordar trabajos dentro del ámbito del arte 3D en la industria del videojuego. Si bien aún quedan aspectos por perfeccionar, como una integración más precisa de lightmaps en el motor, la inclusión de material layering para una personalización realmente premium, el refinamiento del realismo en los materiales o el control más preciso de texel density según los estándares que se aplican en producciones AAA, estos elementos corresponden ya a competencias asociadas a perfiles sénior más experimentados y no suponen una barrera para el desempeño profesional como artista júnior. El objetivo principal era conseguir un conocimiento técnico notable de todas las partes del flujo de trabajo y una visión global de cómo se interrelacionan los procesos, cosa que se cree haber logrado.

Por el contrario, el desconocimiento inicial de ciertos flujos de trabajo implicó una inversión de tiempo considerable no prevista, motivada tanto por la necesidad de realizar múltiples ajustes como por el hecho de rehacer determinadas fases del proceso en etapas avanzadas para estar a la altura de nuevos estándares que se iban conociendo. En retrospectiva, puede que con una visión más clara del alcance técnico se hubiera optado entonces por una propuesta diferente, pero una vez superado el desafío, el resultado es muy positivo.

Con todo, el esfuerzo ha tenido un valor formativo de peso. La experiencia fue, por lo general, entretenida y ha terminado siendo una excelente oportunidad de crecimiento.

Gracias a este trabajo, la distancia entre los conocimientos iniciales y las competencias realmente requeridas en un entorno profesional se ha visto sustancialmente reducida. Durante el periodo de prácticas de empresa quedó claro que aún era necesario reforzar ciertos aspectos técnicos del modelado y texturizado realista para videojuegos, por lo que este proyecto ha sido clave para especializarse. En este sentido, puede considerarse como una transición necesaria para estar mejor preparado para las competencias laborales.

También se espera haber reunido información útil en un mismo lugar, como es en esta guía paso a paso. Se espera de corazón haberla hecho lo suficientemente clara como para que permita replicar un producto de estas características a aquel que, como el autor, tuviera un nivel bajo o intermedio.

En definitiva, este proceso ha sido valioso y necesario, y no hay mejor manera de dejar constancia del viaje personal que dejarlo por escrito.

Capítulo 5. FUTURAS LÍNEAS DE TRABAJO

Una futura línea de investigación es continuar la cadena de montaje de los assets 3D y llevarla al apartado del motor, ya orientado a su integración en un videojuego. En el caso concreto de los props de escenario, gran parte de los procesos posteriores a la creación del modelo están relacionados con el texturizado y su presentación visual en el entorno final.

Subsurface Scattering es un efecto que hace que la luz entre en un material, se esparza por dentro y salga suavemente generando variaciones de rugosidad y color, como pasa con la piel humana o con una vela cuando reciben luz de cerca. Se gestiona mediante añadir mapas de texturas auxiliares en los slots reservados para ello en los motores. También se puede combinar con emisivos. La mayoría de programas de 3D lo soportan. Obtiene una iluminación más realista. En este proyecto no se ha contemplado y sería una buena mejora para algunos materiales específicos.

Para dar variedad a las texturas tileables en un motor gráfico de videojuegos y evitar que un mismo asset se vea repetitivo y sin sustancia, existen las ya mencionadas técnicas de Vertex Painting y Decals.

La primera no ha sido desarrollada en este documento. Por su parte, aunque los decals han sido explorados en parte en este proyecto, no está de más profundizar en su funcionamiento real dentro del motor, donde en lugar de colocarse sobre geometría flotante, se proyectan volumétricamente. Tienen forma de cubo y proyectan sobre lo que encuentren dentro, por capas. Su manejo es simple, pero llevar al extremo la reutilización de assets mediante un uso inteligente de la proyección de decals requiere de un conocimiento técnico considerable. Cabe puntualizar que un decal puede contar con todos los mapas PBR, por lo que se pueden lograr resultados visuales muy potentes, especialmente en motores como Unreal. En el caso de Unity, alcanzar ese mismo nivel de detalle requiere del uso de HDRP o shaders personalizados.

El vertex paint es una técnica para mezclar materiales que es muy útil y está ampliamente extendida. Conviene tenerla en cuenta desde el principio, ya que las mallas deben tener cierta uniformidad y segmentación para soportarla correctamente. Investigarlo nunca está de más, porque combinado con los decals se multiplican las posibilidades de variación optimizada. Vertex paint permite marcar zonas donde un material afecta sobre otro, lo que resulta ideal para generar variaciones basadas en desgaste, suciedad, daño u otras personalizaciones.

Una técnica compleja derivada del vertex paint es el Material Layering, que consiste en crear varias versiones de un mismo material y, mediante vertex paint, ir transicionando entre ellas de forma progresiva y dinámica. Por ejemplo, hacer que un mismo prop pueda pasar entre su versión nueva, oxidada, embarrada, con moho, dañada o con pintura personalizada, ya sea de forma programada o aleatoria. Cada capa tiene su propio conjunto de mapas PBR, y se

mezcla usando vertex color, height maps y weight maps pintados manualmente. Esto permite infinitas combinaciones con un solo shader y unas pocas máscaras, lo cual es ideal para mundos abiertos o escenas con gran variedad visual.

La espina clavada en el desarrollo del proyecto, debido a las limitaciones por dependencias de paquetes en Unity, ha sido no poder indagar al máximo en materiales con simulación de relieve por alteración de coordenadas UV, es decir, mediante Parallax Occlusion Mapping (POM). Aunque los normal maps y el bump son técnicas que consiguen cierto relieve al cambiar la orientación de las normales del modelo, la cantidad de volumen que representan es limitada. El POM genera relieves más pronunciados y convincentes a relativa distancia, sin necesidad de modificar la geometría del modelo, lo que lo hace más ligero que el displacement mapping basado en tessellation, que sí subdivide la malla. Aunque el POM no alcanza el nivel de detalle de un displacement, que su relieve es geometría real, se utiliza con frecuencia en superficies amplias y cercanas al jugador por su buen equilibrio entre realismo y rendimiento. En este producto se ha explorado POM para la versión de Unreal, que lo incluye en algunos de sus materiales, pero se mantuvo a un nivel discreto y no muy elaborado, limitado a unos assets en concreto, con el motivo de no generar discordancias grandes de calidad con la versión de Unity. Una implementación en detalle con muchas más consideraciones de cara a conseguir hiperrealismo es una buena vía para continuar con el proyecto.

Luego, existe una vía para saltarse la mayoría de recomendaciones de este documento acerca de cómo optimizar los assets, que es el uso de “Nanite” en Unreal Engine. Esta tecnología permite mover con facilidad mallas de cantidades absurdas de polígonos, lo que permite modelar todos los detalles y no trabajarlos en la textura. Su uso se está extendiendo gracias al auge de los photo-scans, que es la fotogrametría de objetos reales para convertirlos a modelos tridimensionales. Tanto es así que muchos paquetes de assets de la tienda ya se anuncian como Nanite-dependientes. Eso sí, esta tecnología sólo funciona en gráficas compatibles con DirectX12, por lo que no se puede usar en juegos de móvil o consolas que no sean de última generación (anteriores a PS5/Xbox Series X/S). Se puede investigar cómo diseñar assets directamente para su uso con Nanite.

Por último, se puede investigar la parte de programación relacionada con props, aunque se escape un poco de las competencias habituales del artista 3D. Un buen ejemplo es implementar shaders por código, que es mandatorio si se quieren vender assets de máxima calidad en Unity Asset Store, que no permite dependencias externas como Shader Graph, la herramienta visual para hacer shaders.

También existen flujos para automatizar la instanciación de props por script cuando se van a iterar mucho en escena. Esto consigue colocar assets aleatorizando su escala, rotación, configuraciones de material layers, decals, etc.

A modo de síntesis, las futuras líneas de investigación podrían orientarse hacia:

- Subsurface Scattering para materiales más realistas.
- Vertex Paint para evitar repetición en tileables.
- Decals proyectados en motores para generar variedad y ocultar repetición de texturas.
- Material Layering para combinaciones infinitas.
- Perfeccionar Parallax Occlusion Mapping para falsos relieves realistas.
- Assets para Nanite en Unreal Engine.
- Programación de shaders para no generar dependencias.
- Instanciación dinámica de props en el motor.

Capítulo 6. REFERENCIAS

6.1. Referencias bibliográficas: libros y artículos de investigación

- Ahearn, L. (2017). *3D Game Environment: Create Professional 3D Worlds. 2nd Edition.* Taylor & Francis Group. ISBN 978-1-138-92002-6.
- Ahearn, L. (2009). *3D Game Textures: Create Professional Game Art Using Photoshop. 2nd Edition.* Elsevier. ISBN 978-0-240-81148-2.
- Akeley, K., Foley, J., Feiner, S., Hughes, J.F., McGuire, M., Sklar, D., Van Dam, A. (2013). *Computer Graphics: Principles and Practice.* Addison Wesley Professional. ISBN 978-0321399526.
- Demers, O. (2002). *Digital Texturing & Painting*. Riders Publishing. ISBN 0-7357-0918-1
- Donovan, T. (2010). *Replay: The history of video games*. Yellow Ant. ISBN 978-0-9565072-0-4
- Everly, D. H. (2000). *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. CRC Press. ISBN 978-1558605930
- Gahan, A. (2011). *3dsMax Modelling for Games: Insider's Guide to Game Character, Vehicle, and Environment Modeling*. Elsevier. ISBN 978-0-240-81582-4
- Gregory, J. (2019). *Game Engine Architecture*. Taylor & Francis Group. ISBN 978-1-1380-3545-4
- Kent, S.L. (2001). *The Ultimate History of Video Games: From Pong to Pokemon--The Story Behind the Craze That* Touched *Our Lives and Changed the World*. Crown. ISBN 978-0761536437
- Kent, S.L. (2021). *The Ultimate History of Video Games, Volume 2: Nintendo, Sony, Microsoft, and the Billion-Dollar Battle to Shape Modern Gaming* Crown. ISBN 978-1984825438
- Kumar, A. (2020). *Beginning PBR Texturing: Learn Physically Based Rendering with Allegorithmic's Substance Painter*. Apress. ISBN 978-1-4842-5898-9
- Moioli, G. (2022). *Introduction to Blender 3.0: Learn Organic and Architectural Modeling, Lighting, Materials, Painting, Rendering, and Compositing with Blender*. Apress. ISBN 978-1-4842-7953-3
- Osipa, J. (2010). *Stop Staring: Facial Modelling and Animation Done Right*. Wiley Publishing. ISBN 978-0-470-60990-3

6.2. Otras fuentes

Recursos Web y Contenido Multimedia

- 3Dmodeling Sub-Reddit. (primera publicación 2011, 31 de enero). *3Dprinting*. Reddit. Recuperado el 9 de diciembre de 2024 de https://www.reddit.com/r/3Dmodeling
- Bacquey Habrylo, J. (2025, 10 de marzo). *Mastering Modular & Procedural Techniques For Old West Town In UE5*. 80.lv. https://80.lv/articles/mastering-modular-procedural-techniques-for-old-west-town-in-ue5/
- Benzo. (2024, 23 de abril). *Epic Games lanza Unreal Engine 5.4 con mejoras en la animación y el rendimiento de renderización*. El Otro Lado. https://www.elotrolado.net/noticias/juegos/unreal-engine-5-4-disponible?
- Blender Vitals. (2023, 28 de julio). *Create Curtains in Blender in 1 Minute*. Youtube. https://www.youtube.com/watch?v=DRbA0vgbcJ4&list=WL&index=59
- CG Krab. (2023, 5 de julio). *Export LODs from Blender to UE5*. Youtube. https://www.youtube.com/watch?v=wlVCpgjLE2M&list=WL&index=63
- Coreb Games. (2023, 12 de septiembre). *UE5 | Realistic Textures with Parallax Occlusion Mapping (POM) | Material Tutorial | Unreal Engine 5*. Youtube. https://www.youtube.com/watch?v=K18qfcTFkNw
- Costel, F. (2020). *High Score*. Netflix. https://www.netflix.com/es/title/81019087
- DigiBarn. (2004). *Vintage Computer Festival.* DigiBarn Computer Museum. https://www.digibarn.com/history/04-VCF7-MazeWar/
- Epic Education, Learning and Training. (2022, 7 de noviembre). *Fundamentos del Editor de Unreal: Materiales.* Epic Games Development Forums. https://dev.epicgames.com/community/learning/courses/QPQ/unreal-engine-fundamentos-del-editor-de-unreal-materiales/l2p6/unreal-engine-introduccion-al-renderizado-basado-en-fisicas
- Game Dev Academy. (2022, 24 de octubre). *Improve UE5 Performance with LODs - AUTOMAGICALLY!*. Youtube. https://www.youtube.com/watch?v=NqotFqrRWeo&t=202s
- Heinrichs, J. (2024, 15 de noviembre). *Meshy AI Review: How I Generated 3D Models in One Minute.* Unite.ai. https://www.unite.ai/meshy-ai-review/
- Hu, E. (2024, 3 de enero). *Meshy: Empowering 3D Content Generation.* 80Lv. https://80.lv/articles/meshy-empowering-3d-content-generation
- Jsabbott. (2024, 27 de noviembre). *Quick Start Guide for Materials - Getting Started in Unreal Engine 5*. Epic Games Development Forums. https://dev.epicgames.com/community/learning/tutorials/GP2q/quick-start-guide-for-materials-getting-started-in-unreal-engine-5

- Leartes Studios. (2021, 18 de abril). *Medieval Village Environment* [página de producto]. Fab. Recuperado el 9 de junio de 2025 de https://www.fab.com/listings/177780e9-0c66-42d1-9b03-fa29f8bd5cd0
- Light_Shock. (2022, 10 de abril). *Creating MRAO Textures using free software.* Epic Games Development Forums. https://dev.epicgames.com/community/learning/tutorials/6Gn5/creating-mrao-textures-using-free-software
- McKenzie, T. (2023, 20 de abril). *Art Dump: A Closer Look at Hogwarts Legacy's Props.* 80Lvl. https://80.lv/articles/art-dump-a-closer-look-at-hogwarts-legacy-s-props/
- Mythmatic. (2024, 14 de noviembre). *Roughness Maps Not Working in Unity URP? Here's Why (Full Guide).* Youtube. https://www.youtube.com/watch?v=4OdaWqylA-4 *(Buen vídeo para entender contexto, pero su método de empaquetado puede no funcionar, ver Anexo)*
- Mythmatic. (2024, 8 de diciembre). *How I Create Game-Ready 3D Models 10x Faster (Pro Workflow Secret).* Youtube. https://www.youtube.com/watch?v=4OdaWqylA-4
- Pleyland. (2021, 14 de febrero). *5 Easy Curtains in 10 Minutes - Blender Tutorial*. Youtube. https://www.youtube.com/watch?v=vtwoLRLw9ow&t=420s
- PrismaticaDev. (2024, 21 de junio). *Parallax Occlusion Mapping | 5-Minute Materials [UE5]*. Youtube. https://www.youtube.com/watch?v=h0xytNFiqbQ&t=249s
- Reids Channel. (2020, 7 de febrero). *Unreal Engine 4 - Simple Candle Tutorial*. Youtube. https://www.youtube.com/watch?v=px1QxFr8wzg&t=616s
- Unity Technologies. (2023). *Asset packages: Creating and exporting*. Unity Manual. https://docs.unity3d.com/Manual/AssetPackages.html
- Unity Technologies. (2024, diciembre). *Submission guidelines*. Unity Asset Store. https://assetstore.unity.com/publishing/submission-guidelines
- Woodhouse, F. (2003, August 28). *3D Decals*. GameDev.net. https://www.gamedev.net/tutorials/programming/graphics/3d-decals-r1986/
- Zarus.m. (2022, 24 de diciembre). *Combine different Textures to RGBA Channel Of Single Picture Using Photoshop*. Youtube. https://www.youtube.com/watch?v=0udZKYxtPX4
- ZeusAl. (2013, 9 de julio). *Potencial del nuevo motor gráfico Frostbite 3.* Profesional Review. https://www.profesionalreview.com/2013/07/09/potencial-del-nuevo-motor-grafico-frostbite-3/

Capítulo 7. ANEXOS

Documentación Complementaria

Cómo funciona MeshyAI, la IA que modela en 3D, sus limitaciones actuales, y por qué podría reemplazar la mayoría de puestos de trabajo de 3D para 2030:

https://www.youtube.com/watch?v=-mUPWQf3UU4

Formación adicional sobre texturizado procedural por nodos en Blender:
https://www.youtube.com/@RyanKingArt

Comprobar los requisitos para vender en Fab:
https://dev.epicgames.com/documentation/en-us/fab/asset-file-format-and-structure-requirements-in-fab

Comprobar los requisitos para vender en Unity Asset Store:
https://assetstore.unity.com/publishing/submission-guidelines#1.%20Content%20Restrictions

AutoLOD en Unity:
https://www.youtube.com/watch?v=EqVig88ZC_M&t=268s

Procesos Opcionales

Acerca de modelado y como paso extra recomiendo aprender sobre simulación de físicas para crear props textiles. En este proyecto se crearon manteles y cortinas gracias a ello, en base a planos simulados. Manipular soportes y colisiones para generar comportamientos de pliegue en telas da lugar a assets realistas de forma rápida. Hay recursos formativos acerca de cómo simular físicas en Blender en la bibliografía. Como hace falta subdividir la malla para que funcionen las simulaciones, luego hay que reducir la geometría mediante modificadores “Decimate” o similares si se quiere un polycount viable en assets optimizados para juegos. Es obligatorio extruir el plano con un modificador “Solidify” o manualmente para que constituya un modelo cerrado y se visualice en motores, con mucho cuidado de que no atraviesen caras volteadas. Una vez reducida la geometría del modelo simulado es muy probable que aunque se visualice bien en Blender, dé problemas de sombreado en motores, especialmente en Unreal (no le gusta el suavizado smooth de Blender), por lo que recomiendo crear bordes artesanales para evitar quads mal alineados en las aristas.

Las escenas de prueba en el motor son obligatorias para venta. Es recomendable que tengan una buena iluminación y estén gestionadas debidamente en cada motor. Esto se escapa en cierta medida al alcance del proyecto, puesto que no es de recibo tener una documentación que ocupe 300 páginas. Se recomienda encarecidamente realizar una formación acerca de un funcionamiento básico/intermedio de Unreal y de Unity para que el producto luzca bien.

Otro punto favorable y que añade mucho valor es añadir efectos visuales para props que lo puedan requerir, e incluirlos en prefabs/blueprints ya integrados con el modelo. Por ejemplo, en este proyecto se crearon VFX para la llama de una vela. Se incluyeron efectos de partículas y luces combinados con los modelos de las velas y sus soportes.

Una buena escena ayuda mucho al posicionamiento del producto, ya que las capturas de pantalla y vídeos deben estar renderizados en el motor y no en software 3D externo como Maya o Blender. De lo contrario, se puede considerar como producto engañoso y es muy probable que no pase la revisión manual, y por tanto, no se permita su publicación.

Un controlador de personaje funcional para explorar las escenas es muy buena idea. En el caso de Unreal no es problema porque viene por defecto hecho y funciona directamente. Unity 6 es diferente, y es que tiene dos sistemas de input. Debido a esto, es necesario configurar manualmente el proyecto para usar el antiguo, y especificar en sus opciones qué archivo es el que se usa para las acciones de input si es que se usa el sistema nuevo. Esto impide un funcionamiento directo al importar el paquete, dado que el proyecto al que se carga es individual al cliente y no se puede dejar preparado. La solución profesional es añadir una guía de uso para que se configure tras la importación y funcione, aunque se trate de conocimiento básico del motor. Aún así, se puede pasar por alto el controlador y dejar que se vea el contenido del paquete desde la vista de escena en lugar de en modo “Play”. No se trata de algo obligatorio para que se permita la publicación del asset.

Explicaciones Grabadas en Vídeo

Para algunos de los procesos, a pesar de estar reflejados por escrito en el documento, se consideró que era mejor documentarlos con explicaciones audiovisuales. Por este motivo, algunos flujos de trabajo se han grabado y desgranado en vídeo.

- Cómo exportar de Blender a Unity con pivotes, ejes y orientación correctos, y texturas en estándar metallic-smoothness: https://youtu.be/KV_Fcw3NkKU

  (Este método de empaquetado es más rápido, pero puede perder el canal alfa. Si ocurre, usar el método del vídeo de debajo, que es más robusto, pero aplicándolo al estándar Metallic-Smoothness)

- Texturas y materiales en Unreal: empaquetado MRAOH en Photoshop, opciones de importación, materiales realistas por Parallax Occlusion Mapping, y máscaras de color. https://youtu.be/beK-uCpU5B8