ESNE Escuela Universitaria de Diseño, Innovación y Tecnología

Grado en Diseño y Desarrollo de Videojuegos

Trabajo de Fin de Grado (TFG)
Programación

Optimización del rendimiento del render de mallas transparentes usando el stencil buffer.

Silvia Osoro García

Grupo: Horizon's Tale
Proyecto: A Seed's Tale

Promoción 2017 – 2021

Madrid 2021

D.ª Silvia Oroso García autoriza a que el presente trabajo se guarde y custodie en los repositorios de ESNE Escuela Universitaria de Diseño e Innovación, así como su disposición en abierto del mismo.

Silvia Osoro García | silwia.o.g@gmail.com

Agradecimientos

Gracias en primer lugar a mi familia por haberme apoyado durante todo este camino, y a mis amigos por haber estado ahí conmigo esos días duros durante la carrera.

Gracias al productor/diseñador/diseñador de audio de Horizon's Tale, Jorge, por apoyarme desde el inicio de esta aventura y en cada problema que ha surgido.

Gracias a Manuel Martínez y a Ángel Rodríguez por todo el apoyo, ayuda y tiempo de tutorías que me han servido para saber mucho más de programación y poder realizar la parte de investigación de este proyecto.

Gracias a todos los profesores que me han enseñado y ayudado hasta llegar aquí.

Índice

Índice de figuras

RESUMEN

Esta investigación tiene como propósito entender los conceptos alrededor de las transparencias para aplicar las mejores técnicas de optimización según el problema de transparencias que tengamos, y desarrollar una posible solución al problema de los objetos opacos con transparencias a través del Stencil Buffer.

Vamos a comparar entre tres técnicas y estudiar sus resultados desde distintos ordenadores con distintas gráficas para ver qué diferencias hay, qué nos beneficia de cada una de las técnicas y cuándo usar cada una.

El objetivo con esta investigación es llevar la mejor técnica a un juego en donde la mecánica principal es el cambio de estación. Necesitamos que ciertos objetos desaparezcan y que otros aparezcan de manera dinámica. Lo ideal es que de igual cuántos objetos sean, se quiere que el rendimiento se vea mínimamente afectado por estos cambios, para dejar que el resto del cambio se lleve lo que necesite de rendimiento.
A parte, el deseo de exportar a otras plataformas que no sean PC nos obliga a tener especial cuidado con los recursos que utilizamos.

Palabras clave: Transparencias, Stencil Buffer, Optimización, Render Pipeline, Videogame Development.

ABSTRACT

This research aims to understand the concepts around transparencies to apply the best optimization techniques according to the transparency problem we have, and to develop a possible solution to the problem of opaque objects with transparencies through the Stencil Buffer.

We are going to compare between three techniques and study their results from different computers with different graphs to see what differences there are, what benefits us from each of the techniques and when to use each one.

The goal with this research is to bring the best technique to a game where the main mechanic is the change of season. We need certain objects to disappear and others to appear dynamically. Ideally, no matter how many objects there are, we want the performance to be minimally affected by these changes, to let the rest of the change take what it needs in performance.
In addition, the desire to export to platforms other than PC, forces us to be especially careful with the resources we use.

Keywords: Transparencies, Stencil Buffer, Optimization, Render Pipeline, Video game Development, Unity.

1. Introducción

1.1 Objetivos:

Al desarrollar "A Seed's Tale", pensamos en que queríamos añadir una mecánica que consistiese en **cambiar de estación** a voluntad del jugador. En cada estación pensamos que, en un mismo escenario, habría decoraciones distintas, lo que se traduce en más objetos en escena. Este es el motivo de que necesitásemos que esos objetos se volviesen invisibles y otros visibles de la manera más eficiente y óptima posible. A esto se le añade el deseo de exportar el juego a **otras plataformas** aparte de PC, y esto significa que tenemos que ser mucho más cuidadosos con los recursos del juego.

1.2 Alcance

Con lo anterior en mente, y partiendo de muy pocos conocimientos acerca del stencil buffer y todos los inconvenientes de las transparencias, me propuse investigar técnicas de transparencias y estudiar cuáles son más convenientes según el resultado que queramos conseguir dentro de unos requisitos tales como las limitaciones de las plataformas en las que queremos exportar, o problemas técnicos de las distintas técnicas de transparencias.

2. Marco conceptual y contextual

2.1 Decisiones iniciales

La primera decisión por tomar sería en qué motor desarrollaríamos el proyecto. Los puntos más decisivos para elegir Unity fueron:

- La **experiencia** que tiene el equipo en Unity respecto a otros motores es notablemente mayor, por lo que nos daría mayor agilidad a la hora de encontrar errores, buscar alternativas a lo que queremos hacer si es que las primeras opciones no funcionasen, y en la posibilidad de reutilizar partes hechas en otros proyectos en Unity en este trabajo.

- La Universal Render Pipeline (URP)[1] en Unity nos iba a permitir mejor **rendimiento** con shaders y un **resultado visual** mejor con menos recursos, exportación del proyecto y facilidad de cambiar entre **calidades de imagen**.

- Implementación de **herramientas** que conocíamos exclusivas para Unity que nos iban a facilitar y agilizar tareas: como **Incontrol**[2] para implementar inputs de mandos (también los de Switch con el permiso de Nintendo) a parte del teclado y ratón; **Yarn Spinner**[3] para el sistema de diálogo, **FMOD**[4] aunque también se

[1] https://docs.unity3d.com/Packages/com.unity.render-pipelines.universal@12.0/manual/index.html
[2] http://www.gallantgames.com/pages/incontrol-introduction
[3] https://yarnspinner.dev
[4] https://www.fmod.com

puede usar en otros motores, el diseñador de sonido del equipo, Jorge Aranda López, tenía conocimientos y experiencia con la herramienta en el motor de **Unity**, y aparte, con el **permiso** de Nintendo, también podíamos acceder a FMOD para otras plataformas; y por último, ciertos packs hechos para Unity como el "**The Illustrated Nature Pack**"[5] nos proporcionaban muchos assets y shaders, y con opción para la **URP** en Unity. Además, herramientas como Prefab Painter y Terrain para la isla.

De entre los repositorios que podíamos usar, elegí **GitHub**[6] porque dos de tres integrantes habíamos usado anteriormente ese repositorio (aunque nunca habíamos llevado el control del GitHub) y con la aplicación GitHub Desktop se hacía muy intuitivo y fácil de enseñar y manejar con naturalidad para gente que no había usado nunca o pocas veces repositorios.

[5] https://assetstore.unity.com/packages/3d/vegetation/the-illustrated-nature-153939

[6] GitHub es una plataforma que sirve para alojar proyectos utilizando un sistema de control de versiones.

Optimización del rendimiento del render de mallas transparentes usando el stencil buffer.

Silvia Osoro García
Departamento de programación, ESNE, Comunidad de Madrid

I. Introducción

En el mundo de los videojuegos, podríamos hablar de la optimización como "*el arte de aprovechar al máximo los recursos de un sistema, o mejorar el rendimiento de tu juego*" como bien explica Daniel G. Blázquez. (2006), el porqué de la necesidad de usar técnicas de **optimización** viene de querer lograr cada vez mejor **calidad**, más **objetos**, más **detalles** (partículas, efectos, etc.), es decir, más factores que hacen que el juego necesite más **recursos** que van a hacer que nuestro juego sea menos fluido, pudiendo llegar al extremo de no poder abrir siquiera el juego. Los **recursos** son los tiempos de carga, la utilización de memoria, el tamaño del juego o aplicación y el valor del frame rate o fotogramas por segundo (FPS). De alguna manera tenemos que hacer que todas esas peticiones se cumplan sin perder fluidez del juego, y aquí es donde entra la optimización, que con las técnicas adecuadas nos permitirá aprovechar al máximo los recursos de nuestros ordenadores, consolas y dispositivos.

Aquí vamos a centrarnos en las **transparencias de objetos en videojuegos**. Luego, los objetos es todo aquello que ocupa un volumen o espacio y que esté dentro del juego, y la transparencia es un efecto visual el cual pretende imitar objetos transparentes o invisibles (Alex Dunn, 2014). Más adelante veremos qué problemas pueden dar, pero antes, vamos a ver más conceptos importantes.

Los **objetos** pueden tener materiales, que, hoy en día, los podemos definir como **shader program**, texturas (opcionalmente) y un conjunto de propiedades. Ahora necesitaremos conocer las **texturas**, pues son "*bitmaps", mapas de información o mapas de bits*" (Tokio a, s.f.), es información binaria, la cual proporciona información de color o de otras características (normales, opacidad, reflectividad, etc.). Estas características se "mapean" sobre una superficie 3D que acaba proyectada sobre la pantalla (píxels). Cuantos más píxeles, **más resolución**, lo que se traduce en un mayor **nivel de detalle**. La magia de las texturas viene de que tenemos varias capas de bitmaps en distintos canales para modificar las texturas en tiempo real. Los bitmaps son imágenes o mapas de bits que almacenan la información de los colores que hay en una imagen, guardando así en una imagen de 1 bits el color negro o blanco, 4 bits 16 colores podrán ser representados por pixel, 8 bits 256 colores y 24 bits 16.777.216 tipos de colores para un pixel. Algunos de los tipos de textura más conocidos son Albedo (o Difuso), Especular, Relieve (o Normal map), y Desplazamiento.

Otro concepto que vamos a necesitar conocer es el de los **shaders**. Thomas Denham. (s.f) define los shaders como "*un fragmento de código que se agrega a un renderizado de escena (o cualquier otro método de posproducción)*". Nos cuenta también que, en un principio, los shaders fueron concebidos para alterar el sombreado (shading) de la luz en la escena (de ahí el nombre).
Los shaders son programas que se ejecutan en la **GPU** para aplicar transformaciones a vértices y pixels. Se pueden usar para calcular iluminación, pero también otras características que condicionan el aspecto de un material (por ejemplo, el borde / contorno negro en cel shading). Los shaders procesan la información de vértices y texturas en la GPU para, finalmente producir el color de cada píxel, que luego devuelve la información de iluminación y color, es decir, los shaders trabajan con la pipeline (flujo de trabajo) referente a gráficos y da la información de cómo renderizar cada pixel.

White, S., Coulter, D., Batchelor, D., Satran, M. (2018) nos explican que el **Stencil Buffer** es una técnica en la cual se pone una máscara a los pixeles de una imagen para controlar si el píxel se dibuja o no. Lo más sencillo que se puede hacer con el Stencil Buffer sería limitar las zonas que vamos a renderizar (así como en la técnica del estarcido). Luego tenemos otros usos más avanzados en el que aprovechan los lazos entre el Stencil Buffer y el **Z-buffer** (maneja las coordenadas de profundidad de imágenes) en el proceso de renderizado. Hay gran variedad de **combinaciones** entre técnicas con el Stencil Buffer para conseguir muchos **tipos de efectos**, como Shadow Volume (volumen de las sombras), desvanecimiento, siluetas, dibujo de contorno, Two-Sided Stencil.
A pesar de que podemos lograr acabados como el del Stencil Buffer mediante otro tipo de shaders, una de las ventajas más grandes que nos aporta esta técnica es que empieza a ejecutarse en la fase temprana del proceso de renderizado, actuando como un early-out (de salida temprana), evitando más partes de procesos intensivos, como los shaders. **Previene** que el procesamiento se fragmente de más, lo que es una **ventaja** del rendimiento.

Necesitamos objetos con transparencia ya sea por darle más **realismo** a ciertos objetos del juego o por **detalles** que queremos cuidar dentro de éste, como, por ejemplo, hacer transparente un objeto que está entre la cámara y el jugador para poder ver a través de él, desvanecer objetos, y mucho más.

Ya hemos nombrado las **Render Pipelines**, pero vamos a definirlo a grandes rasgos, ya que hay mucho más detrás de ellas. Cuando nos referimos a la Render pipeline o Tubería de renderizado, estamos hablando del **flujo de trabajo** que es necesario para la representación de gráficos basado en un tipo de hardware de gráficos (tarjeta gráfica). Funciona normalmente recibiendo una **escena** en 3D de entrada para generar lo que vemos por pantalla, una imagen en dos dimensiones. Las **etapas** de una render pipeline es: **transformar** los modelos 3D (traslación o rotación); **iluminar** los modelos (fuentes de luz, reflejo) aunque es más común que en este paso procesen la iluminación solo en los vértices de los polígonos de los modelos para luego interpolarlos en el proceso de rasterización; **transformación** de vista para ver la escena desde la perspectiva de la cámara; **proyección** y transformación donde la geometría captada por la cámara es transformada en un espacio de **imagen 2D**; **clipping** que aun no siendo necesario para recibir correctamente la salida, acelera el proceso descartando aquellos modelos o geometrías que están fuera de la visión de la cámara, lo que se traduce en no rasterizar y postprocesar lo que no aparecerá en la imagen final; el **shading** que texturiza cada fragmento individual (prepíxeles); **rasterización**, proceso en el que convertimos la imagen 2D

en un formato de mapa de bits en donde se determina el valor de cada píxel resultante; y por último, la **visualización**, cuando tenemos los píxeles ya visibles.

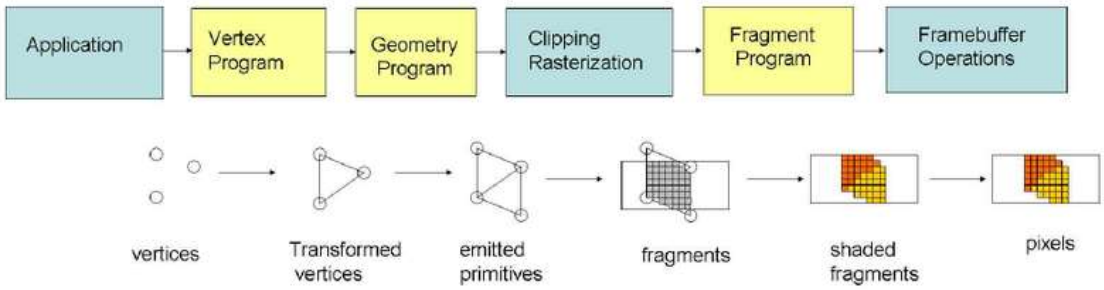

Figura 1 (s.f) Pipeline de gráficos en OpenGL [Figura]

Recuperado de https://www.researchgate.net/profile/Christoph-Guetter/publication/235696712/figure/fig1/AS:299742132228097@1448475501091/The-graphics-pipeline-in-OpenGL-consists-of-these-5-steps-in-the-new-generation-of-cards.png

Otro dato importante es que, si tenemos objetos transparentes, debemos tener en cuenta sobre qué se va a superponer, por lo que primero deberá dibujar todos los objetos opacos, ordenar los objetos transparentes por distancia a la cámara (del más lejano al más cercano) y dibujarlos.

II. ¿Por qué no debo abusar de los materiales transparentes?

Sikachev, P. (s.f) nos cuenta varios **problemas** que nos pueden surgir, el primero, porque tener **muchos** objetos transparentes va a afectar el **rendimiento**, sobre todo cuando estos objetos están dibujados sobre otros muchas veces. A esto se le conoce como **overdraw**, o sobre dibujado, es decir, cuando el mismo píxel se dibuja varias veces. Otro de los problemas es que no todos los objetos transparentes son **compatibles** con todas las técnicas de renderizado, por ejemplo, en motores de **renderizado diferido**, hay que pasarlos de manera especial. El renderizado diferido es el proceso en el que posiciones, normales y materiales se renderizan en el buffer de **geometría** (G-buffer) antes de sombrearlo todo. Y el último problema que vamos a tratar es el **orden de renderizado**, y es que cuando tenemos objetos juntos, quien nos ayuda a separarlos es el Z-buffer con la profundidad, y al final los que "más importan" son los que están más cerca, cualquier cosa detrás no merece la pena renderizarla, a no ser que el objeto que tengas más cerca sea transparente, es por ello que necesitaremos un renderizado eficiente que conozca unas **prioridades** ya vistas (primero los objetos opacos, luego los transparentes). De esta manera quitamos la mayoría de los comportamientos extraños, pero el problema viene que cuando tu objeto usa un material transparente, da igual que el 99% de tu objeto sea opaco, para el renderizado, tu objeto es transparente, por lo que se va a dibujar según su profundidad con relación a los demás objetos transparentes de la escena. Si hay muchos objetos transparentes van a surgir problemas en el momento en el que se intersecan sus mallas.

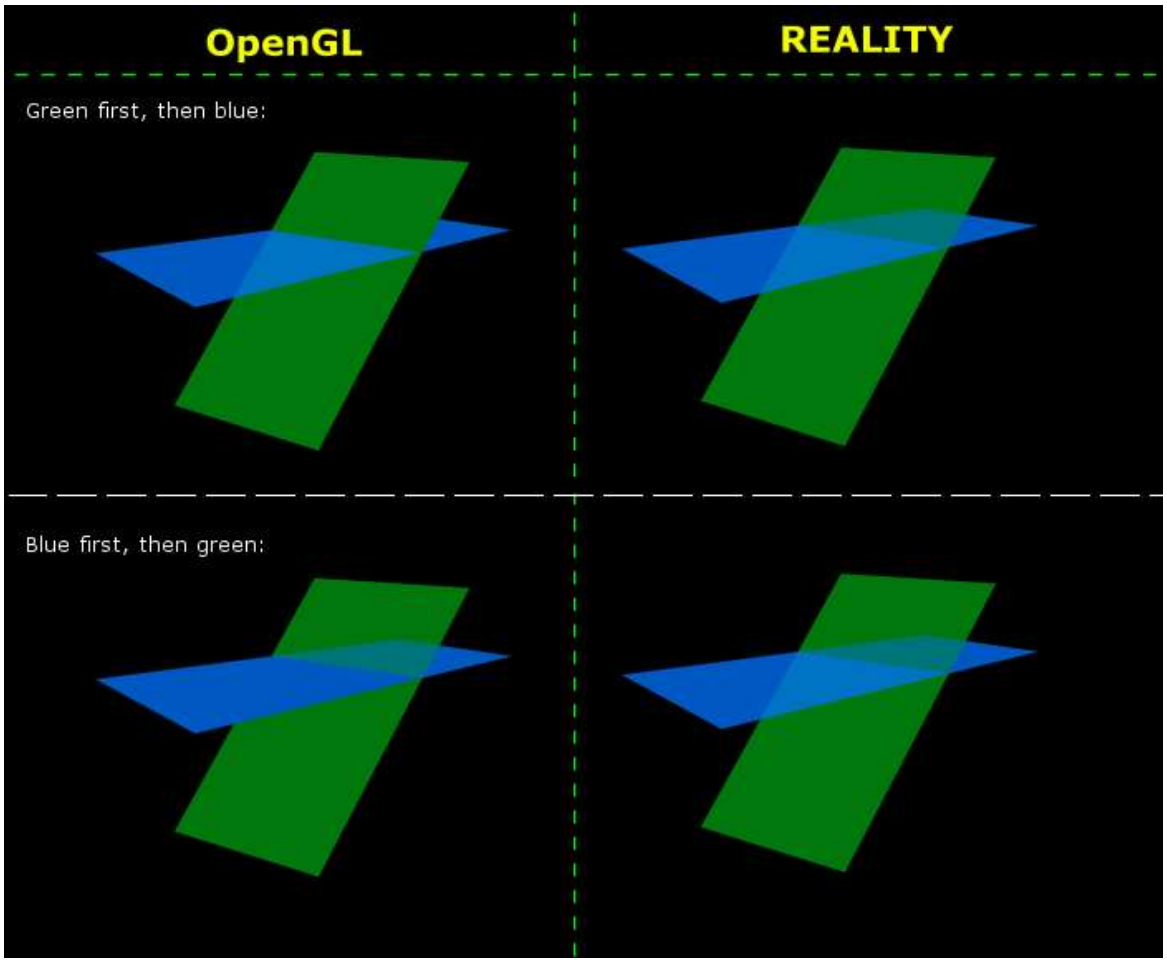

Figura 2 (s. f.) Ejemplo de intersección entre mallas. OpenGL vs Realidad. [Figura].

Recuperado de https://i.stack.imgur.com/ImsJl.png

OpenGL es una interfaz de programación (API) o biblioteca de código que nos sirve para escribir aplicaciones que produzcan gráficos 2D y 3D.

También podemos añadir a estos problemas el que, en el proceso de renderización, los objetos se van **renderizando por grupos**, y los últimos y más **costosos** son los objetos transparentes. El problema aquí es que viajan todos en la misma tubería, por lo que cuantos más objetos de ese tipo haya, más tardará en terminar de renderizar.
Y es por estas cosas que en sistemas para **VR**[7] se hace inviable meter objetos transparentes. En **móvil** y **consolas** deberemos reducir al máximo el número de objetos transparentes también.

Estos son los problemas más significativos con los objetos transparentes, y veremos qué métodos son los más recomendables para distintos usos de los objetos transparentes.

[7] La realidad virtual (RV) es un entorno de escenas u objetos de apariencia real. La acepción más común refiere a un entorno generado mediante tecnología informática, que crea en el usuario la sensación de estar inmerso en él. Dicho entorno es contemplado por el usuario a través de un dispositivo conocido como gafas o casco de realidad virtual.
https://es.wikipedia.org/wiki/Realidad_virtual#Definición

III. Casos de optimización de transparencias

En este apartado iremos planteando situaciones en las cuales queramos meter algún tipo de transparencia y cómo y por qué debemos de usar ciertas técnicas.

Stencil buffer:

El stencil buffer, como ya hemos visto antes, se puede usar para enmascarar píxels, controlando qué píxel va a ser dibujado o no. Lo bueno de las técnicas con stencil buffer es que nos sirven para controlar de manera muy precisa todo aquello que queramos renderizar en pantalla, además, desde una etapa temprana ya hace su función, por lo que los siguientes pasos no tendrán que tratar los píxels que el stencil buffer ya ha descartado, por lo que ya va a ser más óptimo.

Vamos a ver más de cerca las técnicas más usadas del stencil buffer:

- **Compositing**: con esta técnica seleccionamos un área, la cual enmascaramos, y metemos el render de la escena que queramos **espejar**. Aunque no tiene tanto que ver con nuestras transparencias, si no con **reflejos**, no está de más conocerlo.

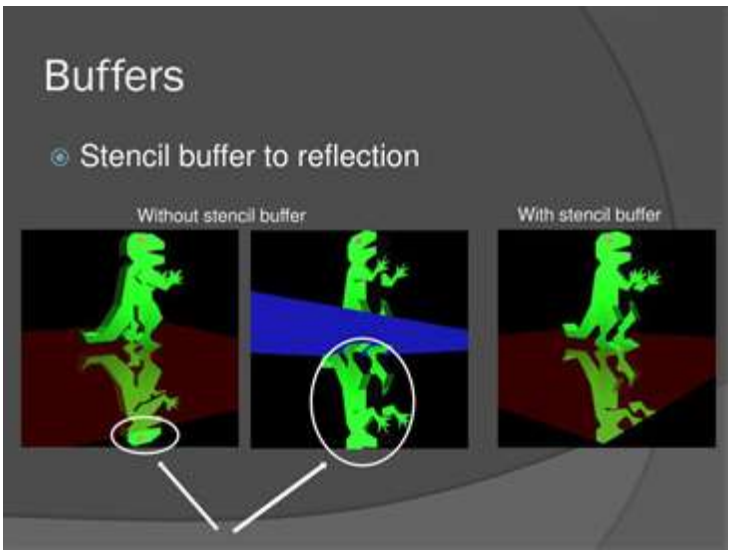

Figura 3 Tate, K. (2019) Comparación de reflejos de distintas técnicas vs con Stencil Buffer. [Figura].

Recuperado de https://slideplayer.com/slide/12976139/

- **Decaling**: esta técnica es útil en algunas circunstancias para no usar el **z-buffering**. Consiste en que, si hay dos imágenes en el mismo plano, van a tener problemas de "**z-fighting**"[8], por lo que las imágenes estarán intentando verse a la vez. Con esta técnica lo que hacemos es enmascarar la zona de la imagen que irá detrás, en la cual se va a colocar la imagen que vaya a estar delante. De esta manera conseguimos que la imagen esté delante de otra, estando en el mismo plano y sin depender del z-buffering.
  Esto lo podríamos hacer con el **blending** de múltiples texturas, pero esta técnica limitaría el número de efectos que puede hacer nuestra aplicación, por lo que usando el stencil buffer y aplicando la técnica de decaling, dejamos **espacio libre** para otros efectos, ya que el stencil buffer se aplica en una etapa muy temprana del renderizado, y no se junta con otros procesos más costosos como lo haría el blending de múltiples texturas.
  El pequeño inconveniente que tiene el z-buffering es que, a pesar de que se ejecuta rápido, usa algo de memoria, y con esto podríamos evitar usarlo. Hoy en día el consumo de memoria de un Z-Buffer es insignificante en proporción a la memoria que tienen las GPUs.
  ¿**Por qué usar esta técnica en vez de recurrir al z-buffering**? Porque el z-buffering no puede hacer superficies transparentes si no es que le añadimos código adicional y para solucionar el conflicto de z-fighting.
  Aun así, el z-buffering tiene como **pros** que es sencillo de implementar y se ejecuta relativamente rápido.

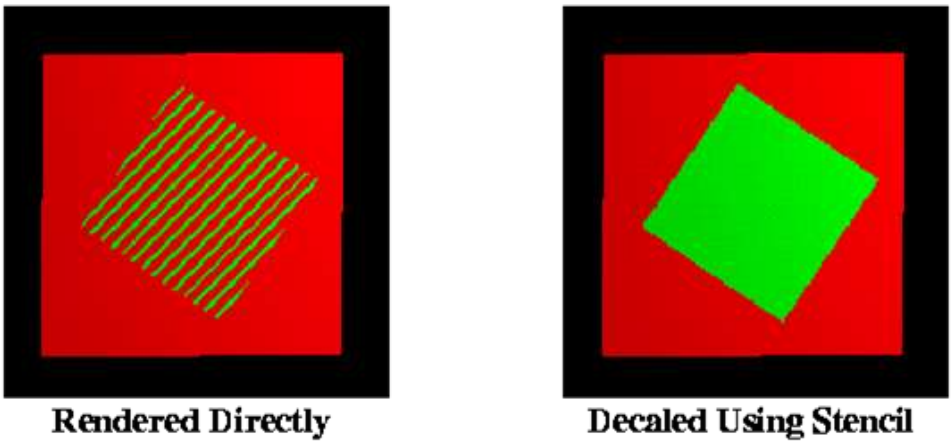

Figura 4 (s.f) Uso de Stencil Buffer para renderizar polígonos coplanarios. [Figura].

Recuperado de
https://www.opengl.org/archives/resources/code/samples/advanced/advanced97/notes/node198.html

[8] Interferencia entre texturas, también llamada lucha Z o lucha de planos, es un fenómeno del renderizado 3D que se produce cuando dos o más primitivas tienen distancias muy similares a la cámara. En consecuencia, sus valores son similares o idénticos en el búfer z, que indica la profundidad de los elementos. https://es.wikipedia.org/wiki/Interferencia_entre_texturas

- **Dissolves**: esta técnica recrea efectos muy interesantes, ya que una imagen se va reemplazando por otra. Donde entra aquí el stencil buffer es en que renderizamos dos imágenes distintas, y el stencil buffer controla qué píxels de cada imagen se va a dibujar. Podemos definir la "**plantilla**" **enmascarada** (que definirá cómo queremos que resulte la disolución) y copiarla al buffer en fotogramas de manera incremental. También podemos definir una máscara como plantilla base para el primer fotograma e ir alternándola de manera incremental.

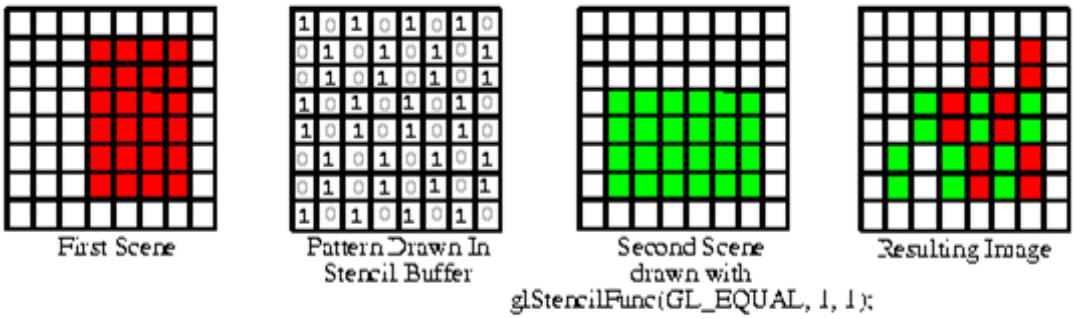

Figura 5 (s.f) Uso de stencil para disolver entre imágenes. [Figura].

Recuperado de https://www.opengl.org/archives/resources/code/samples/advanced/advanced97/notes/node197.html#:~:text=The%20stencil%20buffer%20can%20be,%2C%20smoothing%2C%20and%20other%20effects.

  - Fades: esta técnica es prácticamente como la anterior, de hecho, se considera como un caso especial de disolución, con la variación de que se pasa de negro o blanco al render de una escena (**fade in**) o pasamos de una escena a negro o blanco (**fade out**). La magia de esta técnica es lo dicho anteriormente, podemos usar el patrón que queramos (plantilla).

  - Swipes: al igual que en el dissolve, esta técnica necesita definir las máscaras que vaya a cargar el stencil, para dibujar los píxels de la imagen final y dejar de dibujar los de la imagen inicial. Es algo más complejo porque los píxels de la imagen final se leen en el orden contrario al efecto de swipe que queremos hacer. Si queremos que sea de izquierda a derecha, la aplicación deberá leer la imagen final de derecha a izquierda

Figura 6 Kramer, B. (2016) Ejemplo Fade. [Figura].

Recuperado de https://forum.defold.com/t/screen-fade-using-stencil-buffers/2678/2

- **Outlines and Silhouettes**: esta técnica sirve para crear una línea que rodea el contorno del objeto. Tendríamos que aplicar una máscara a un objeto que es de la misma forma, pero algo más grande (que el objeto), teniendo como resultado una imagen que sería el outline, y por último se rellena ese outline con un color sólido.
  Si la máscara del stencil es del mismo tamaño y forma del objeto al que queremos aplicar la técnica, el espacio en el que tendría que estar el objeto original, como hemos hecho para el outline, estaría relleno de negro para formar la silueta del objeto. Este es un dato muy importante porque aplicando unos cálculos a la **silueta** resultante, podemos imitar el comportamiento de las sombras en tiempo real. Nos es muy útil porque debemos tener en cuenta que cuando hacemos un objeto invisible con nuestro stencil buffer, el objeto realmente sigue ahí, por lo que nos seguirá apareciendo su sombra, y si le hacemos agujeros, no se verán reflejados en ellos. Los autores Shahrizal Sunar, M. y Kolivand, H. enseñan en un documento, llamado *To Combine Silhouette Detection and Stencil Buffer for Generating Real-Time Shadow*[9], cómo crear, de la mejor manera posible, sombras en tiempo real a partir de siluetas hechas con Stencil Buffer.

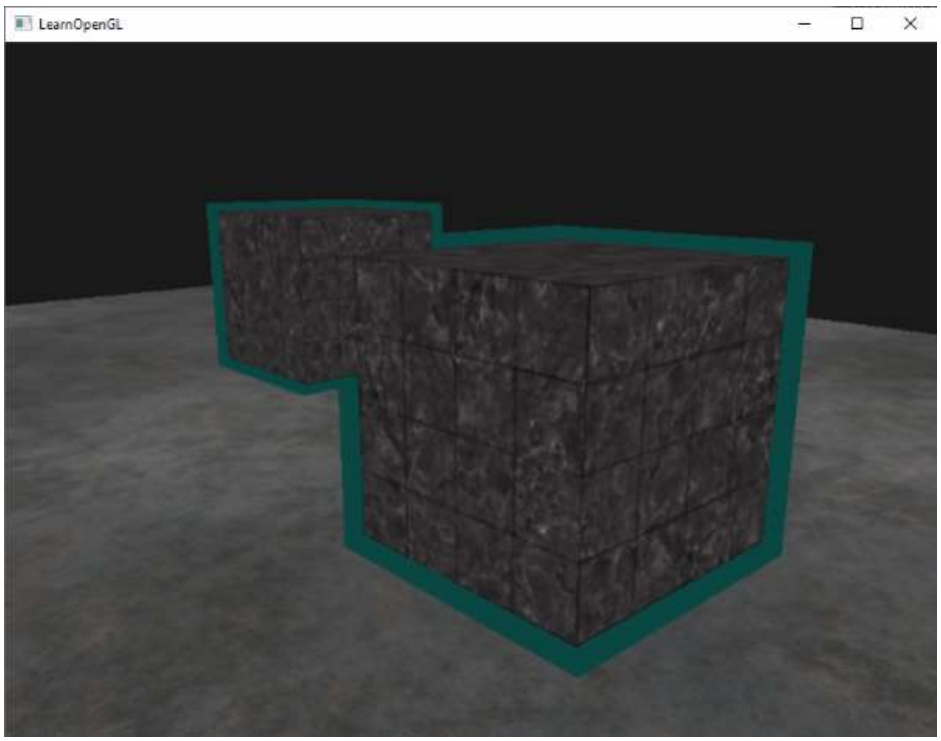

Figura 7 DeVries, J. Ejemplo Outline. [Figura].

Recuperado de https://learnopengl.com/Advanced-OpenGL/Stencil-testing

9 https://www.researchgate.net/publication/236592288_To_Combine_Silhouette_Detection_and_Stencil_Buffer_for_Generating_Real-Time_Shadow

- **Two-sided Stencil**: esta técnica surge de mejorar una anterior, los Shadow Volumes, que consistían en, usando el **occluding geometry** (te quedas con las formas geométricas que capta la imagen de la cámara e ignoras las de atrás que no se ven) sacaba la silueta de los bordes y los alargaba en dirección contraria a la luz y formando un conjunto de volúmenes en 3D. El problema de esta técnica es que tenía que **renderizar dos veces esos volúmenes** en el stencil buffer, renderizando primero los polígonos que daban hacia delante, y un segundo render para los polígonos que daban hacia atrás, por lo que en cuanto tengamos muchos volúmenes y encima varias luces, se hace inviable renderizar sombras. Con la técnica del two-sided stencil, se dibuja en una **única pasada por volumen de sombra, por luz**.

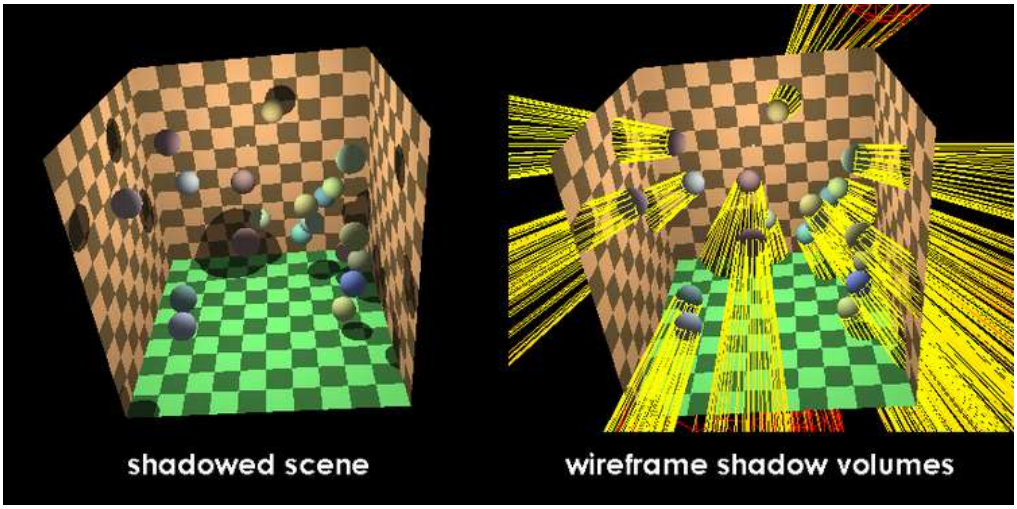

Figura 8 R. (2006). Demostración de los volúmenes de sombra con Stencil Buffer. [Ilustración].

Recuperado de https://en.wikipedia.org/wiki/Shadow_volume

Los contras del stencil buffer es que de por sí solo sirven para las transparencias totalmente invisibles. Para el resto de las transparencias nos iremos a otras técnicas. Otro de los inconvenientes es que en pantalla no veremos el objeto, pero éste sigue ahí, por lo que no dejará pasar, por ejemplo, luz, a no ser que nos metamos en calcularle las sombras, que, aunque es totalmente viable, requiere de más tiempo de desarrollo que usando lo que ya tiene nuestro motor, a pesar de que cueste más, esto sería una pelea entre tiempo que se le puede dedicar y cuánto podemos sacrificar de rendimiento.

Cutout:

Ahora vamos a ver la técnica de cutout, que es un caso especial de las transparencias. Lo que le diferencia de ellas es que los problemas de ordenación que suelen tener los objetos transparentes no ocurren con esta técnica; el shader es enteramente opaco o transparente, y nos sirve para cosas como cadenas, césped, árboles, objetos con agujeros, etc.
¿Qué ventajas tenemos con este tipo de shader? Como no es un tipo de shader transparente del todo, ya que es tratado de otra manera, no va a tener los problemas que nos generaban los transparentes, como por ejemplo, si teníamos una parte opaca y otra transparente, este shader lo considerará enteramente opaco y se hará cargo de que la zona transparente desaparezca, según las combinaciones que hay con este shader, se llevará a cabo de una manera u otra, y aunque se usa la técnica del stencil buffer de elegir qué zona se renderiza y qué zona no, la gran mayoría usan el canal alpha en vez de descartarlo. Aparte, los objetos que usan esta

técnica pueden castear y recibir sombras, ya que usan el canal alpha. Para añadir, es una técnica bastante barata, mucho más que los objetos transparentes.
Las desventajas que podríamos sacar es que no sería más óptimo que el stencil buffer ya que usa el canal alpha.

Figura 9 R. (2012). Ejemplo de cutout. [Figura].

Recuperado de https://www.youtube.com/watch?v=s3RKGAj9Uzk

Order independent transparency (OIT)

Esta técnica nos va a ayudar con los problemas de intersección entre transparencias si es que no tenemos otra opción que mantenerlas.
OIT es el resultado de técnicas de rasterización para renderizar transparencias en escenas 3D. No necesita que la geometría esté ordenada para componer la escena.

Cuando renderizamos una escena con transparencias, éstas las renderizamos mezclando todas las superficies en un solo buffer. El orden en el que se mezcla afectará a cómo se verá cada superficie, por lo que se renderiza desde el más lejano al más cercano o viceversa, pero el hecho de ordenarlos para renderizarlos puede costar bastante tiempo, y a veces siquiera devuelve una solución, como, por ejemplo, cuando la geometría interseca entre sí o hay superposición circular. Aparte, la implementación es algo compleja.

Pues lo que propone la técnica OIT es ordenar la geometría por píxel, después rasterización. Para ser más exactos, necesitaremos **guardar todos los fragmentos** antes de ordenar y componer la escena, esto multiplica los requisitos de memoria, empeora el rendimiento y aumenta la complejidad, por lo que introduce diversos problemas nuevos que hay que tener en cuenta.

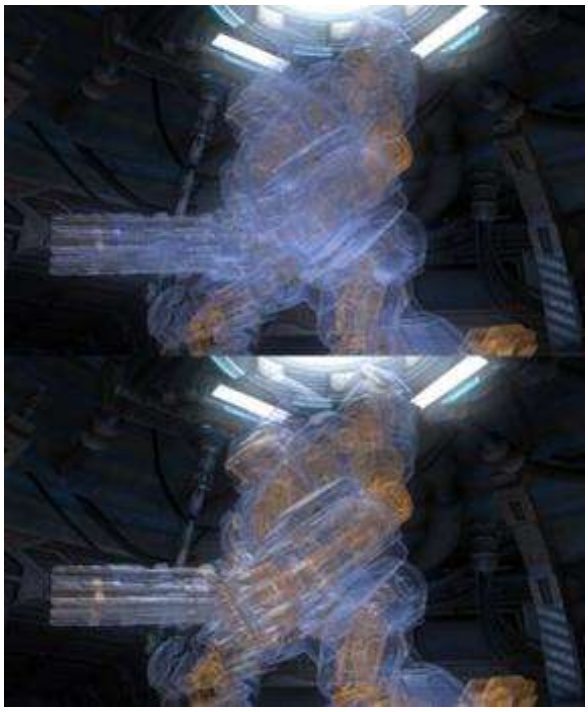

Figura 10 AMD (2017). Comparación entre la técnica de alpha blending vs OIT. [Figura].

Recuperado de https://en.wikipedia.org/wiki/Order-independent_transparency

IV. Pruebas

Por lo que hemos visto, debiéramos evitar usar objetos transparentes, ya que sacrifican mucho rendimiento. Aún así, hemos visto que ciertas técnicas pueden no servirnos para cada caso individual o no nos aporta tal mejora de rendimiento como para sustituir las que estábamos usando.

Ahora vamos a comprobar si las pruebas que haremos encajan con la parte teórica vista y las pretensiones de demostrar que el Stencil Buffer sería un gran sustituto de ciertos tipos de transparencias tales como disoluciones, transiciones, etc.

Empezamos con un proyecto de Unity en la Universal Render Pipeline.
Vamos a empezar con el Stencil Buffer:

Técnicas:

Stencil Buffer

En la URP se necesita un proceso distinto. Para ello seguí páginas[10] que explicaban el cambio que tiene la URP de las demás pipelines.
Primero hay que comentar que la herramienta Stencil Graph no nos deja crear un shader para configurar nuestro Stencil Buffer, si no que tenemos que acceder a él a través del Renderer de la URP.

[10] https://theslidefactory.com/see-through-objects-with-stencil-buffers-using-unity-urp/

Creé un Shader Unlit con esto:

```
Shader "Custom/Mask"
{
    Properties{}

        SubShader{

            Tags {
                "RenderType" = "Opaque"
                }

            Pass {
                ZWrite Off
                }
    }
}
```

Figura 11 Shader Unlit Máscara

Marca que el objeto se renderizará con otros shaders opacos y el **ZWrite Off** indica que no renderiza ningún píxel, lo que hará que nuestra máscara no será visible.
Tras esto, creé 3 layers[11] en Unity: **Mask** para la máscara, **SeeThrough** para los objetos que queramos ocultar, y **ReverseMask** para que a través de la máscara aparezca un objeto. La layer de ReverseMask, si todo sale como esperado, la usaré en el proyecto final para hacer que aparezcan objetos de la siguiente estación mientras oculto las de la layer de SeeThrough (anterior estación).
Configuré el Renderer del proyecto (Universal Render Pipeline Asset_Renderer) y desmarqué esas mismas layers que creé en Filtering (Ambas Layer Mask).

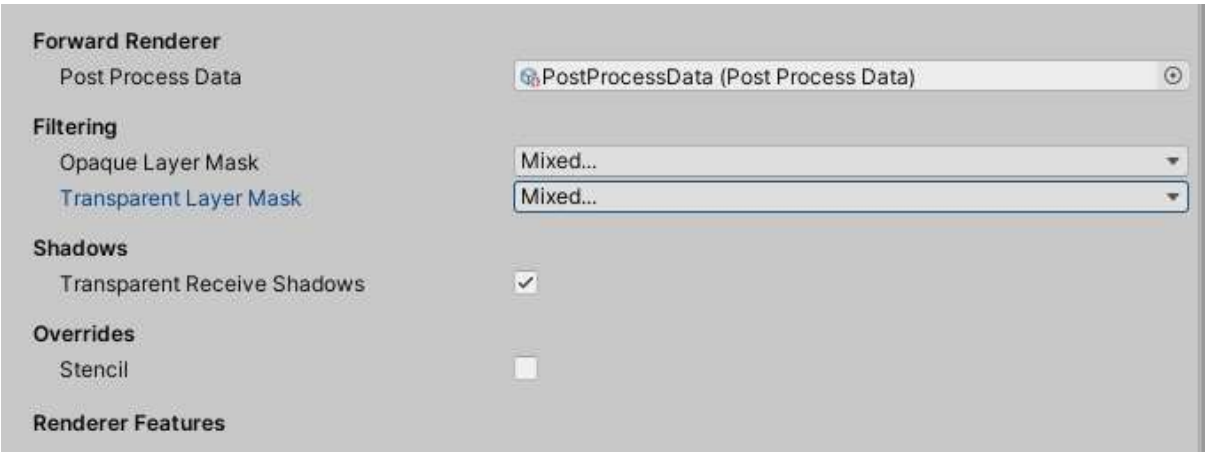

Figura 12 Configuración de layers para el Stencil Buffer en la URP

[11] https://docs.unity3d.com/Manual/Layers.html

Siguió el añadir Render Feature, **Render Objects**. Cada vez que creaba una me daba un error, pero cuando guardaba la escena ya funcionaba mal, por lo que debe ser algún tipo de bug.
Primero, la **Mask** queremos que siempre se esté ejecutando el Replace, es decir, se escribirá el valor en el buffer y dentro de la mesh de la máscara, tendremos el valor de Stencil a 1.

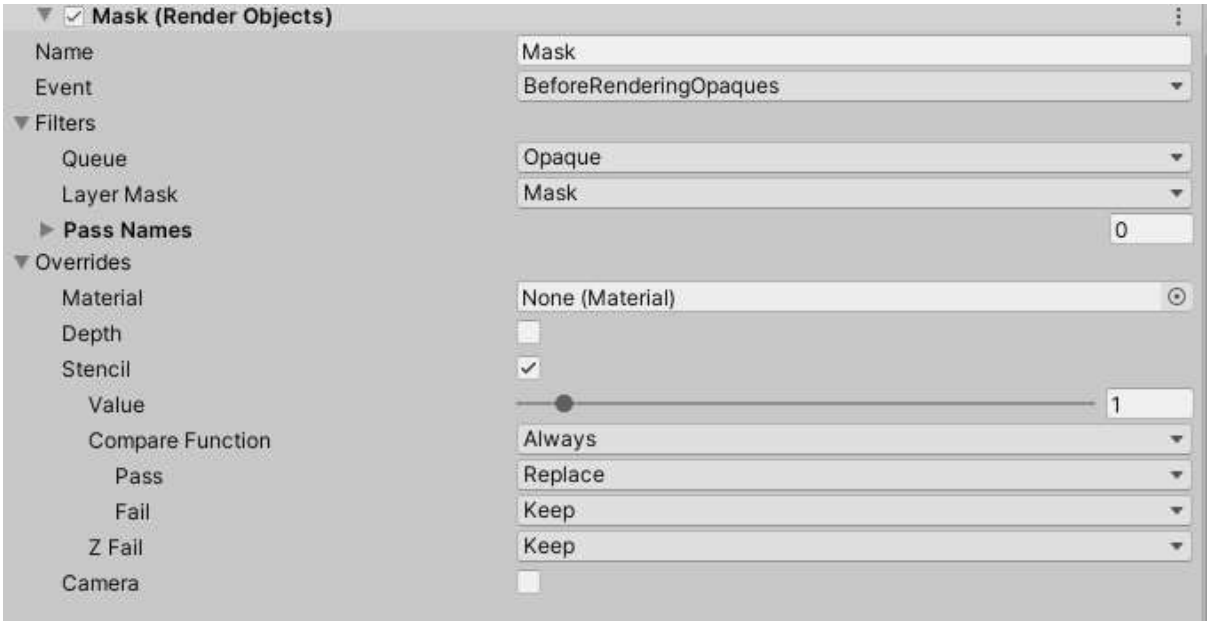

Figura 13 Configuración de la capa Mask en el Renderer de Unity

Para la layer **SeeThrough**, si el valor de referencia difiere con el valor actual en el Stencil Buffer renderizará los pixels de esa layer con un valor de Stencil distinto, es decir, distinto de 1, esto es porque ya la capa es 1, por lo que todo lo que esté **detrás** de la máscara, no se renderizará.

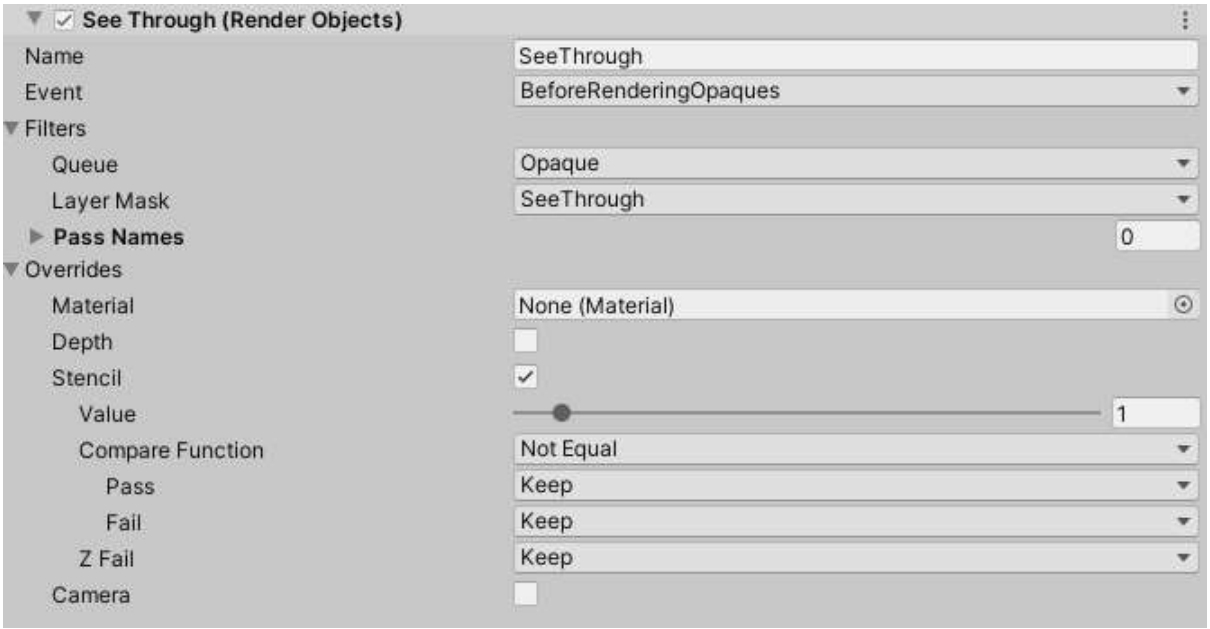

Figura 14 Configuración de la capa SeeThrough en el Renderer de Unity

Por último, la layer de **ReverseMask** mira si su valor de referencia es menor que el valor actual del Stencil Buffer para escribir ese valor de referencia en el buffer. Si no es así, mantiene el valor que ya estaba.

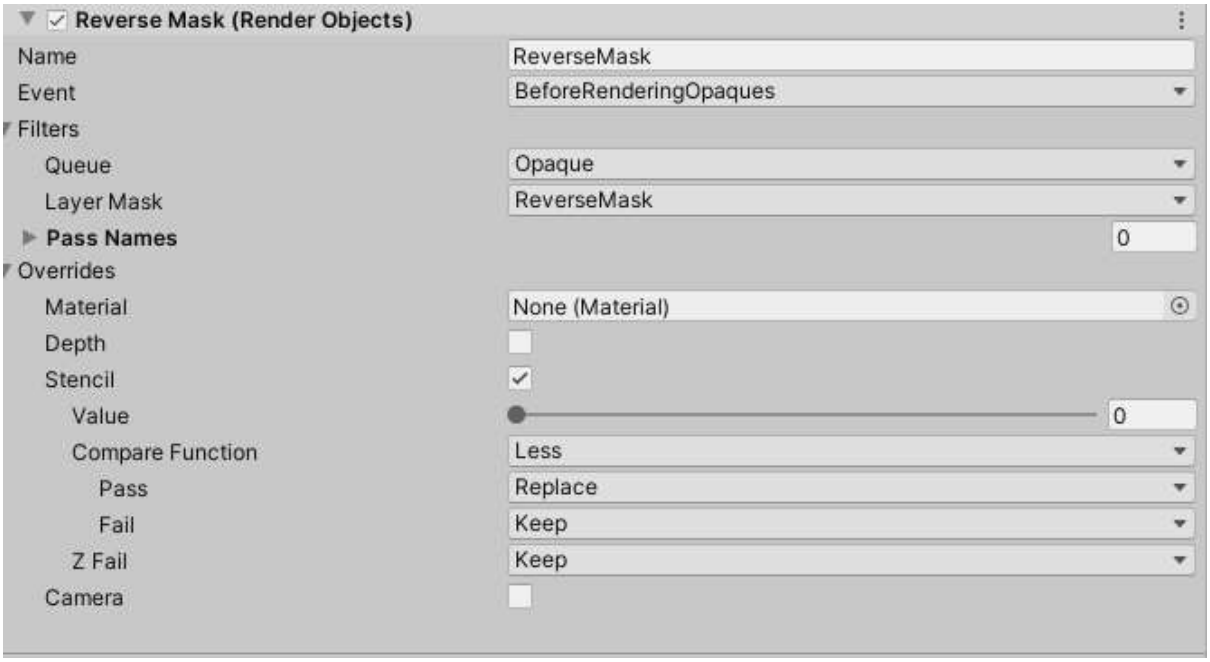

Figura 15 Configuración de la capa RenderMask en el Renderer de Unity

Usé un objeto con el shader hecho anteriormente y lo coloqué en la layer de Mask. A los objetos que quería que les afectase esta máscara, les puse la de SeeThrough. Todos los objetos debían ser **opacos**.

Finalmente, cuando la esfera Mask va aumentando su tamaño, va ocultando los objetos a su paso.

Alpha

Esta parte no fue tan rebuscada. En base a unos shaders Lit[12] del propio Unity URP y que están aplicados a los objetos y cambiarles el tipo de superficie a **Transparent** y asegurar que el Blending Mode es **Alpha**. Aquí no tuve que hacer uso de layers específicas.

[12] https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@7.1/manual/Lit-Shader.html

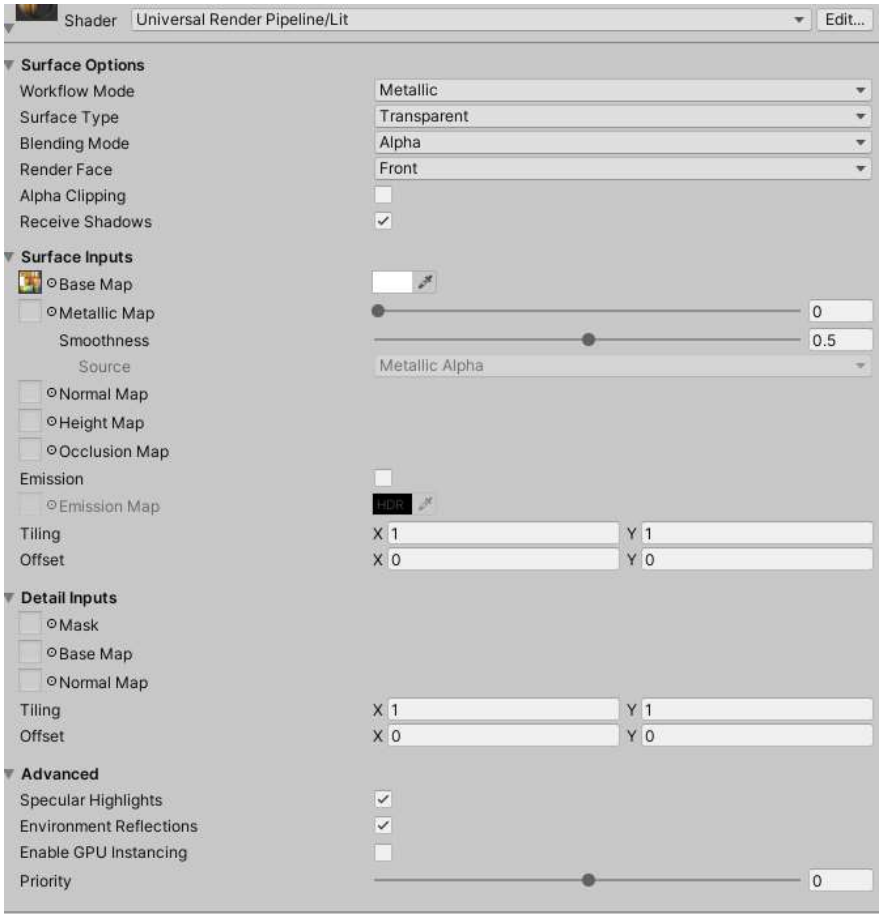

Figura 16 Configuración del shader transparente

Según iba haciendo colisión la Mask (que ya no tiene ni el shader ni layer Mask, pero nos seguiremos refiriendo así a nuestra esfera) los objetos iban cambiando su Alpha hasta llegar a 0.

Cutout

Esta técnica requería de un **shader con superficie transparente**, por lo menos esta versión simple del shader:

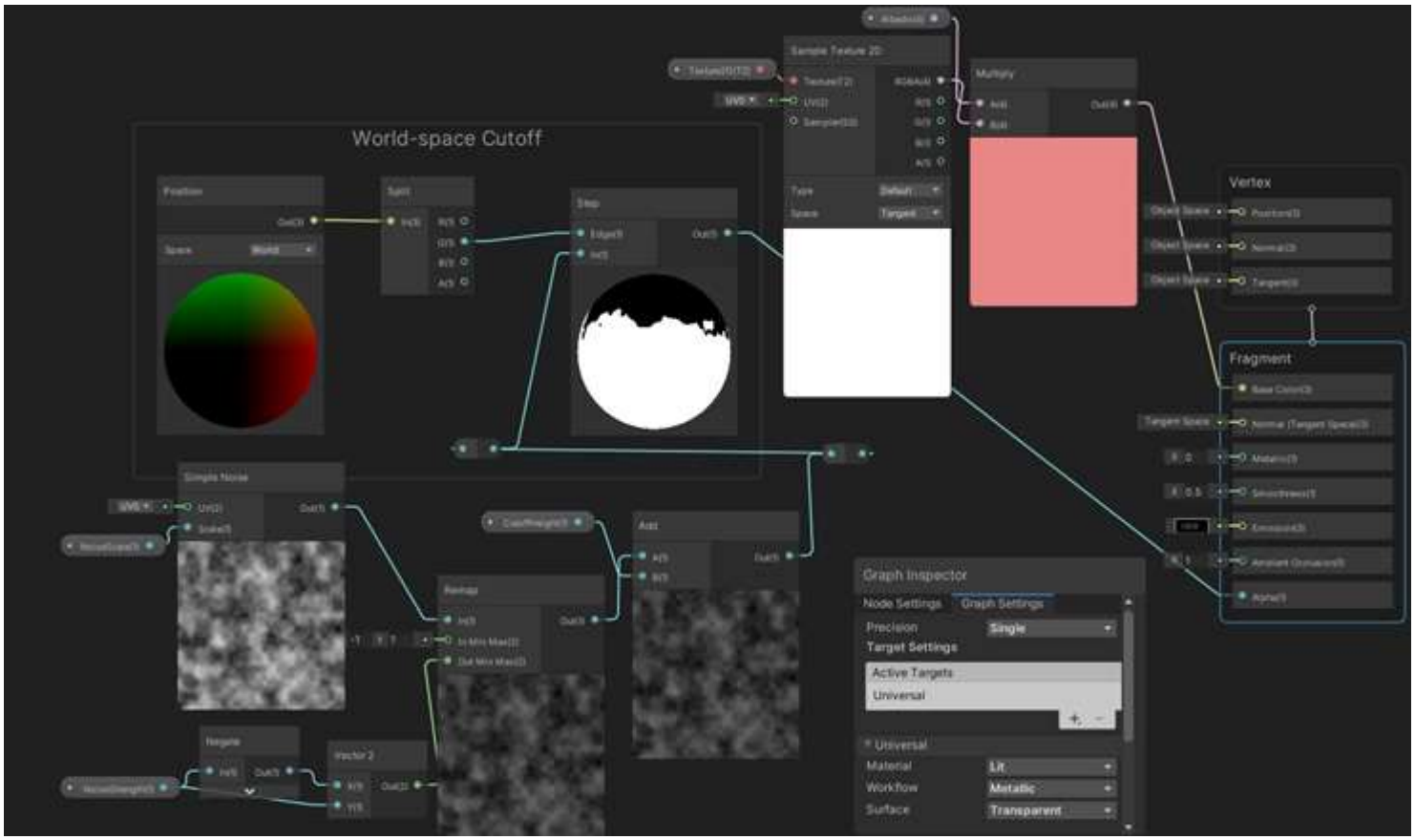

Figura 17 Cutout Shader

Con esto preparado para el shader, según aumente la variable **CutoffHeight** se le añadirá más ruido[13], que irá haciendo invisible esas partes afectadas por el Cutoff hasta que se expanda por todo el objeto.

Esa variable se irá modificando con el tiempo e irá incrementando, haciendo que los objetos vayan desapareciendo con el ruido. Esto se activa una vez que el objeto entra en colisión con nuestro Mask (igual que el de Alpha).

[13] https://www.haroldserrano.com/blog/noise-in-computer-graphics-a-brief-introduction

```
        else if(test == Behaviour.CutOut_Test)
        {
            var time = Time.time * Mathf.PI * 0.25f;

            float height = renderer.material.GetFloat("_CutoffHeight");
            height -= time * (objectHeight / 50.0f);
            SetHeight(height);
        }

    }

}

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.layer == 0)
    {
        startFading = true;
    }
}

private void SetHeight(float height)
{
    renderer.material.SetFloat("_CutoffHeight", height);
    renderer.material.SetFloat("_NoiseStrength", noiseStrength);
}
```

Figura 18 Código que acompaña al shader Cutout

Una vez tenemos todo el material para hacer las pruebas, procedemos con las escenas:

Escenas

Por cada técnica hice dos escenas:

- Una escena con un solo objeto que tiene 69,000,000 triángulos. En cada escena le aplico la técnica correspondiente con los pasos mencionados anteriormente.

Figura 19 Escena con objeto

- Una escena con varios objetos de distintas formas en estilo poly y misma textura, descargada de la Asset Store. Le aplico los pasos de cada técnica a todos los objetos.

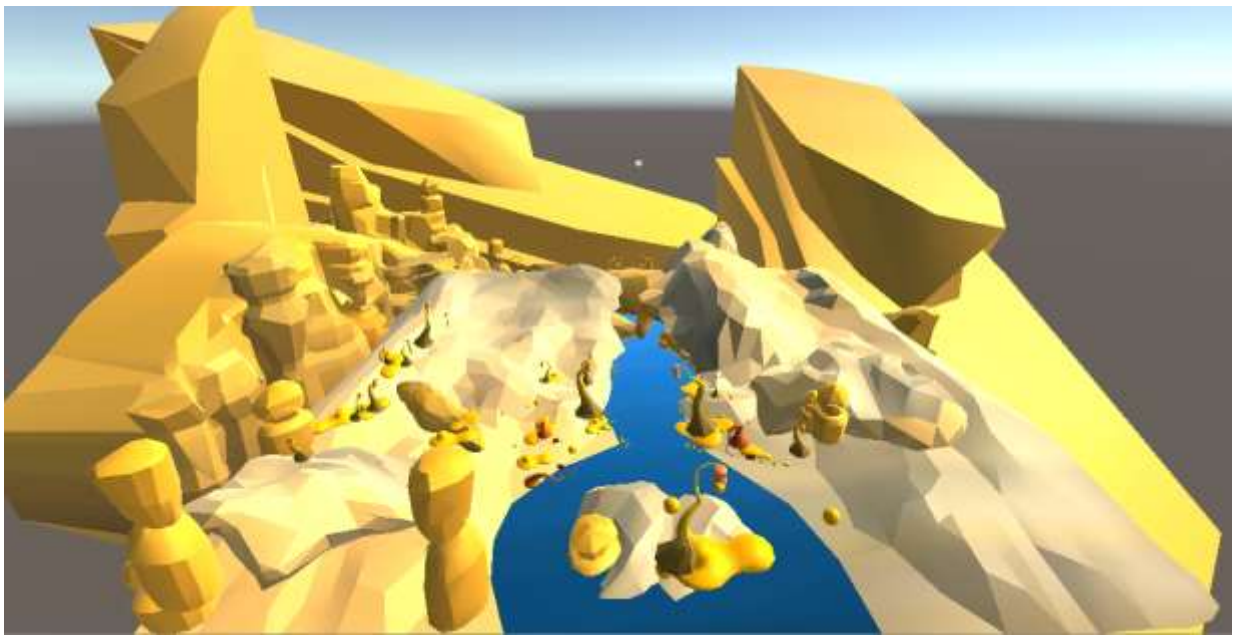

Figura 20 Escena maquetada

Antes de hacer build de cada una, era necesario desactivar el VSyncCount[14] del Quality Settings para que el límite de FPS no lo limitase en 60 y así poder medir cada escena.

[14] https://docs.unity3d.com/ScriptReference/QualitySettings-vSyncCount.html

Datos

Para sacar los datos he usado la aplicación Fraps y he sacado los datos de cada escena en tres ordenadores con distintas características. Cada prueba ha comenzado desde que empieza la escena (después de que termine de cargar lo que necesite cada escena) y terminan cuando todos los objetos de la escena están completamente en invisible, algunas técnicas tardan más que otras simplemente por la escena, no por la técnica, por eso no compararemos tiempos entre ellas.
Entre escenas vamos a comparar principalmente los mínimos y máximos FPS, y entre cada gráfica, el tiempo que tarda cada una en conseguir que todos los objetos estén invisibles, aparte de hacer alguna apreciación entre los FPS mínimos y máximos de cada escena en cada una de ellas.
Las gráficas que tenemos son NVIDIA GeForce GTX 1660 Ti, ASUS Radeon RX 470, AMD Radeon HD 7500M/7600M.
A mayor número de FPS, mejor es el rendimiento (es un dato que lo refleja).

Primero voy a dejar los datos extraídos y luego veremos sus gráficas comparativas:

NVIDIA GeForce GTX 1660 Ti

- Alpha
    - Object: Frames: 1105 - Time: 4015ms - Avg: 275.218 - Min: 270 - Max: 278
    - Scene: Frames: 9735 - Time: 17547ms - Avg: 554.796 - Min: 418 - Max: 787

- Cutout
    - Object: Frames: 3285 - Time: 12828ms - Avg: 256.080 - Min: 252 - Max: 263
    - Scene: Frames: 8703 - Time: 13016ms - Avg: 668.639 - Min: 488 - Max: 909

- Stencil Buffer
    - Object: Frames: 13417 - Time: 50016ms - Avg: 268.254 - Min: 263 - Max: 280
    - Scene: Frames: 20184 - Time: 14031ms - Avg: 1438.529 - Min: 1321 - Max: 1546

ASUS Radeon RX 470

- Alpha
    - Object: Frames: 923 - Time: 6063ms - Avg: 152.235 - Min: 147 - Max: 161
    - Scene: Frames: 5474 - Time: 17625ms - Avg: 310.582 - Min: 218 - Max: 472

- Cutout
    - Object: Frames: 2724 - Time: 16922ms - Avg: 160.974 - Min: 159 - Max: 163
    - Scene: Frames: 5726 - Time: 13110ms - Avg: 436.766 - Min: 314 - Max: 595

- Stencil Buffer
    - Object: Frames: 7616 - Time: 48422ms - Avg: 157.284 - Min: 156 - Max: 159
    - Scene: Frames: 14410 - Time: 13922ms - Avg: 1035.052 - Min: 894 - Max: 1199

AMD Radeon HD 7500M/7600M

- Alpha
  - Object: Frames: 1008 - Time: 86344ms - Avg: 11.674 - Min: 11 - Max: 13
  - Scene: Frames: 2022 - Time: 29531ms - Avg: 68.470 - Min: 54 - Max: 112

- Cutout
  - Object: Frames: 583 - Time: 87141ms - Avg: 6.690 - Min: 5 - Max: 8
  - Scene: Frames: 1692 - Time: 15765ms - Avg: 107.326 - Min: 81 - Max: 143

- Stencil Buffer
  - Object: Frames: 536 - Time: 45547ms - Avg: 11.768 - Min: 11 - Max: 13
  - Scene: Frames: 2595 - Time: 13875ms - Avg: 187.027 - Min: 150 - Max: 216

Gráficas

Vamos a separarlos por el grupo de escenas:

- Objeto con muchos triángulos:

Comparativa FPS de cada técnica agrupados por cada gráfica.

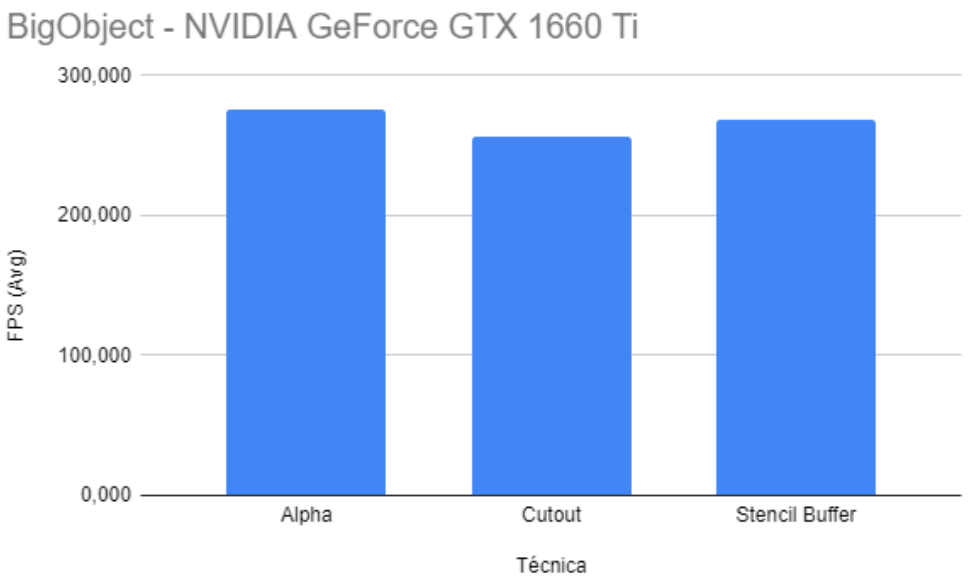

Figura 21 Gráfico Objeto grande con NVIDIA GeForce GTX 1660 Ti. Comparativa FPS (Avg) vs Técnicas

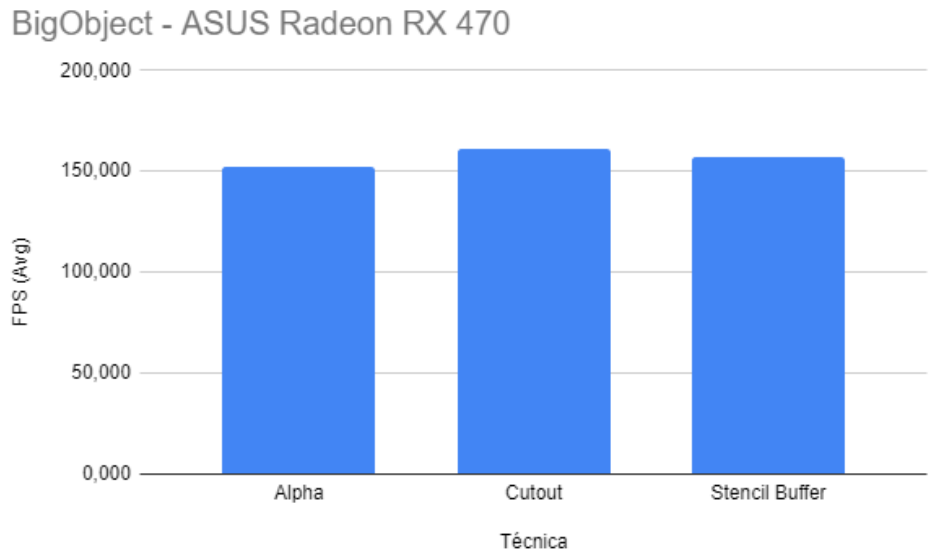

Figura 22 Gráfico Objeto grande con ASUS Radeon RX 470. Comparativa FPS (Avg) vs Técnicas

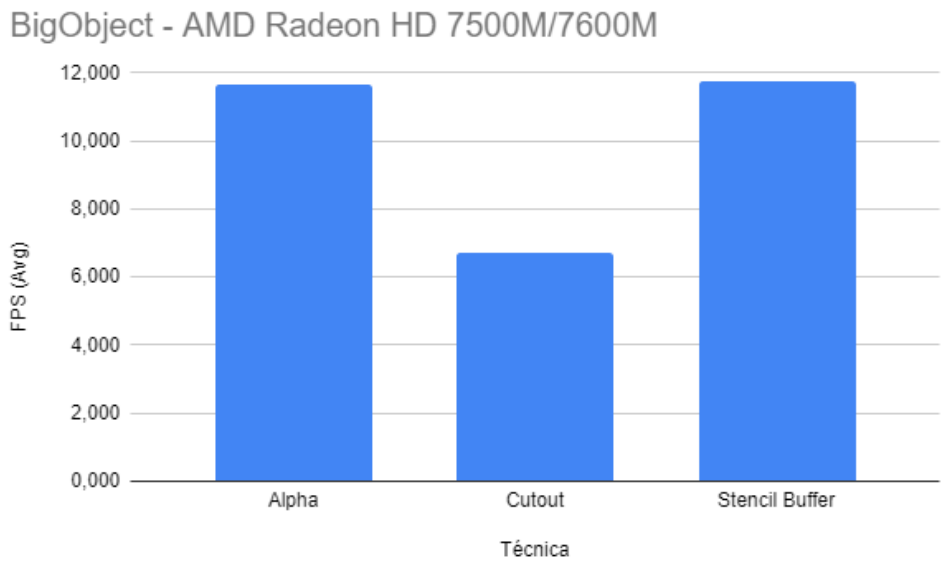

Figura 23 Gráfico Objeto grande con AMD Radeon HD 7500M/7600M. Comparativa FPS (Avg) vs Técnicas

Comparativa tiempos de cada gráfica agrupados por cada técnica.

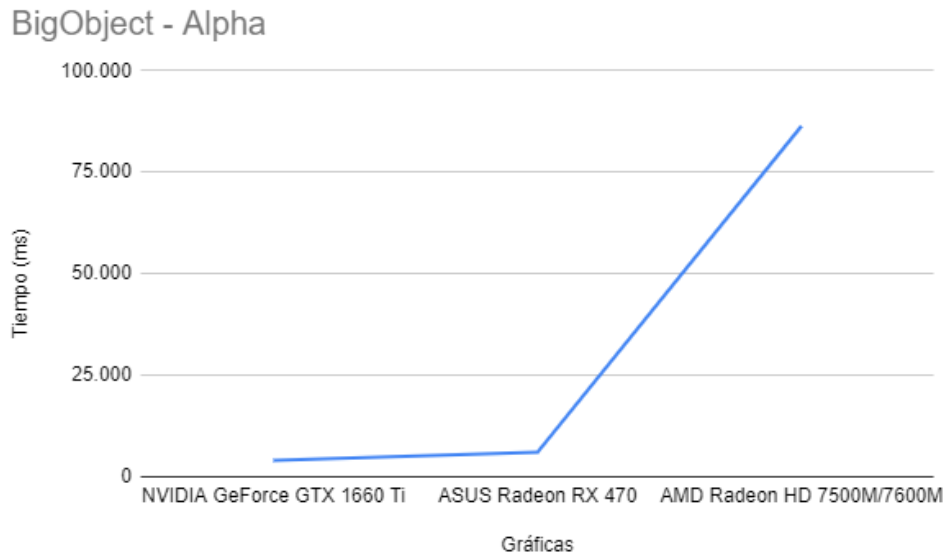

Figura 24 Gráfico Objeto grande con técnica Alpha. Tiempo(ms) vs gráficas

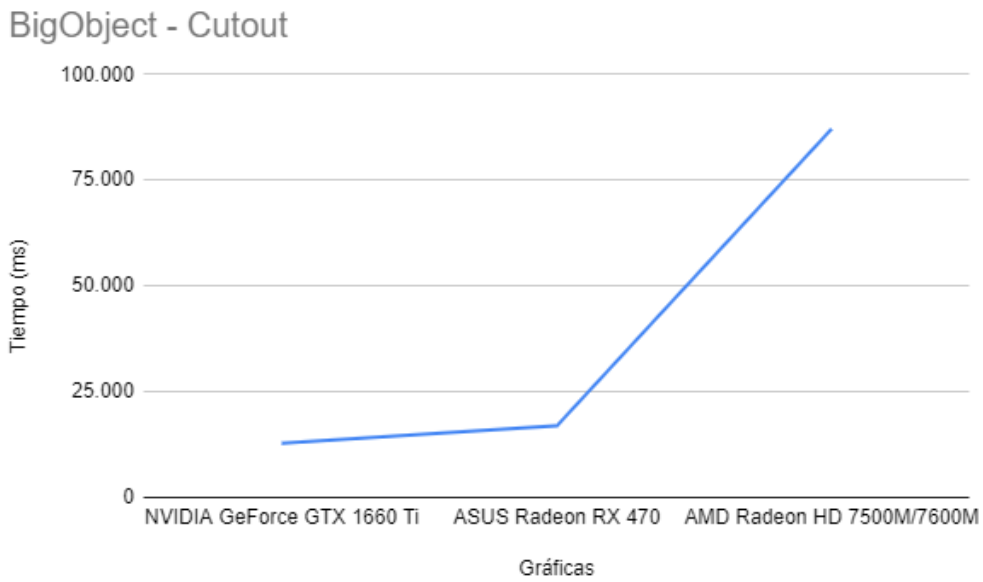

Figura 25 Gráfico Objeto grande con técnica Cutout. Tiempo(ms) vs gráficas

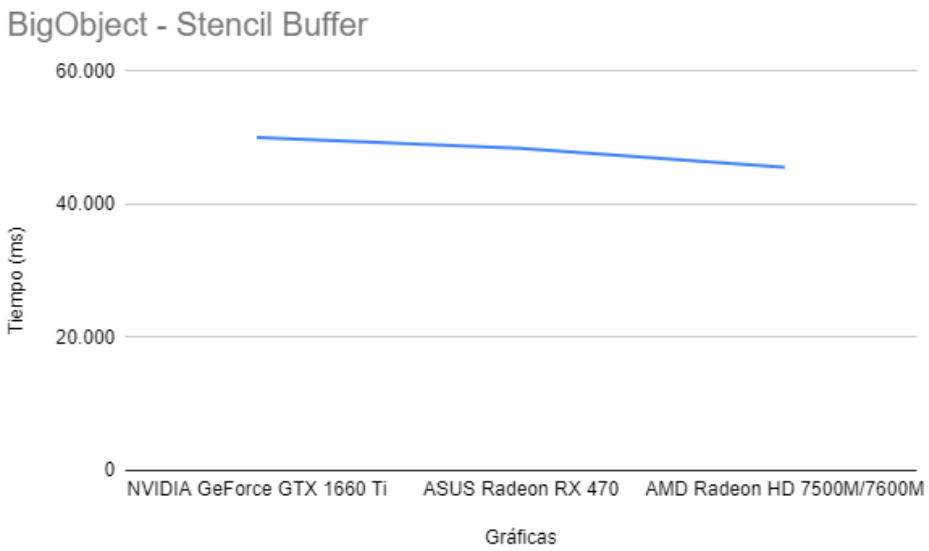

Figura 26 Gráfico Objeto grande con técnica Stencil Buffer. Tiempo(ms) vs gráficas

- Escena con varios objetos:
  Comparativa FPS de cada técnica agrupados por cada gráfica.

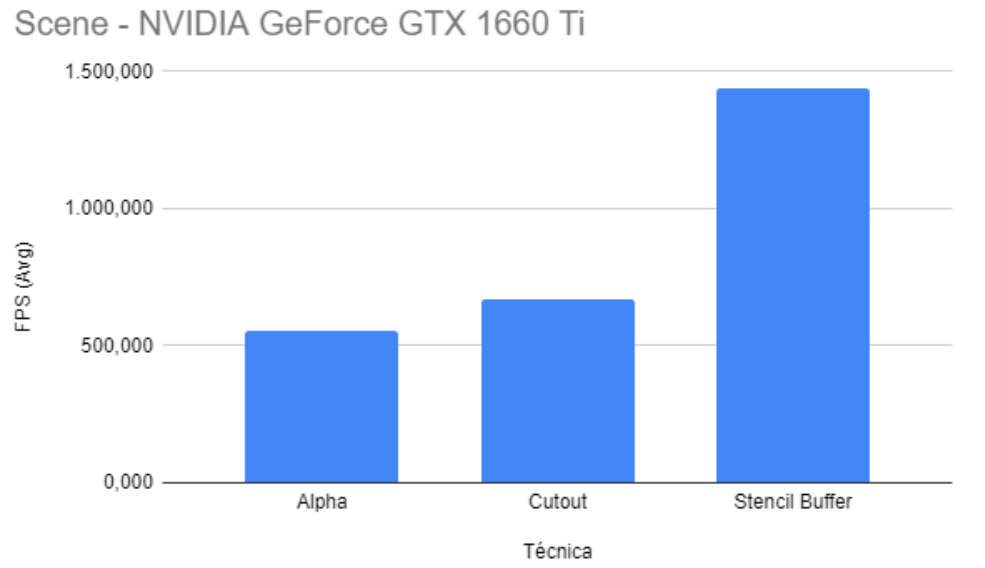

Figura 27 Gráfico Escena con NVIDIA GeForce GTX 1660 Ti. FPS(Avg) vs técnicas

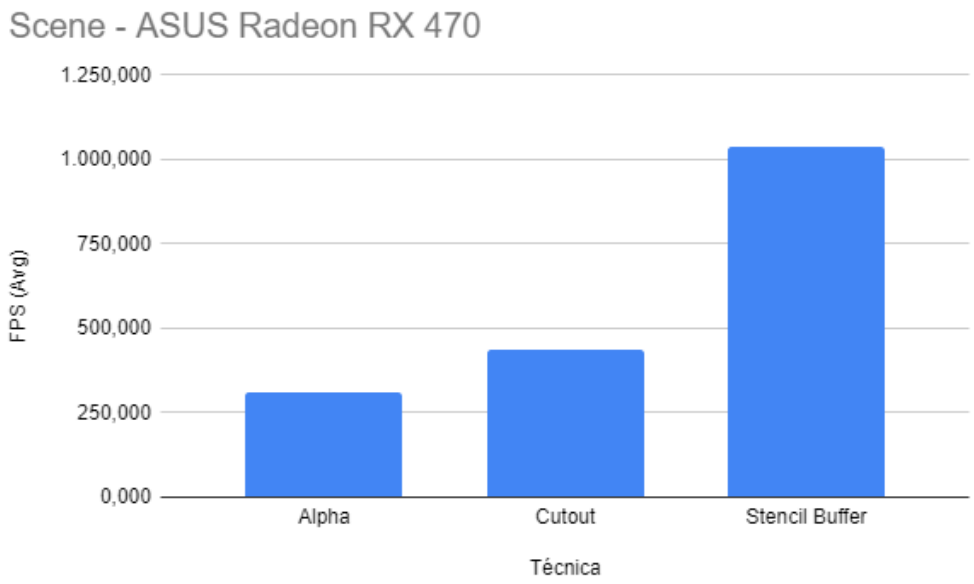

Figura 28 Gráfico Escena con ASUS Radeon RX 470. FPS(Avg) vs técnicas

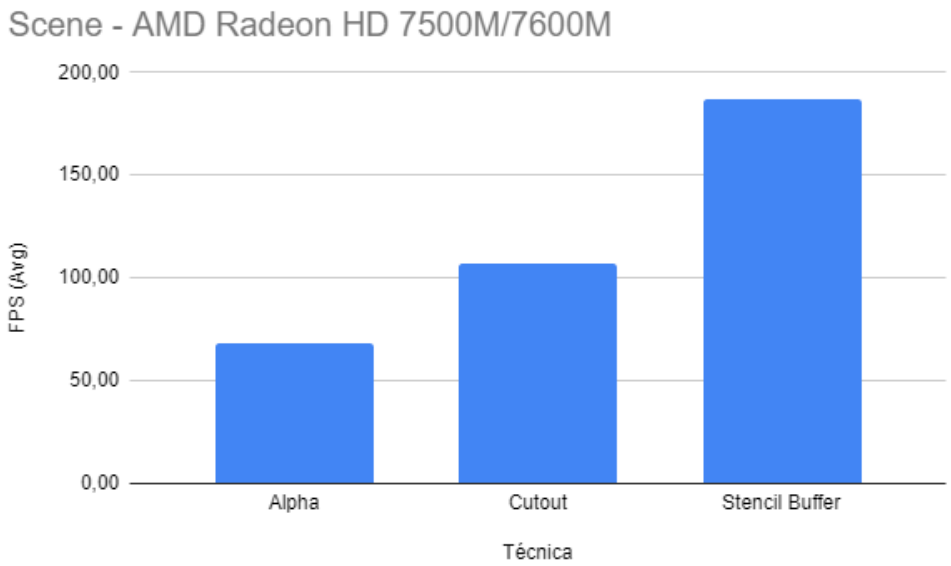

Figura 29 Gráfico Escena con AMD Radeon HD 7500M/7600M. FPS(Avg) vs técnicas

Comparativa tiempos de cada gráfica agrupados por cada técnica.

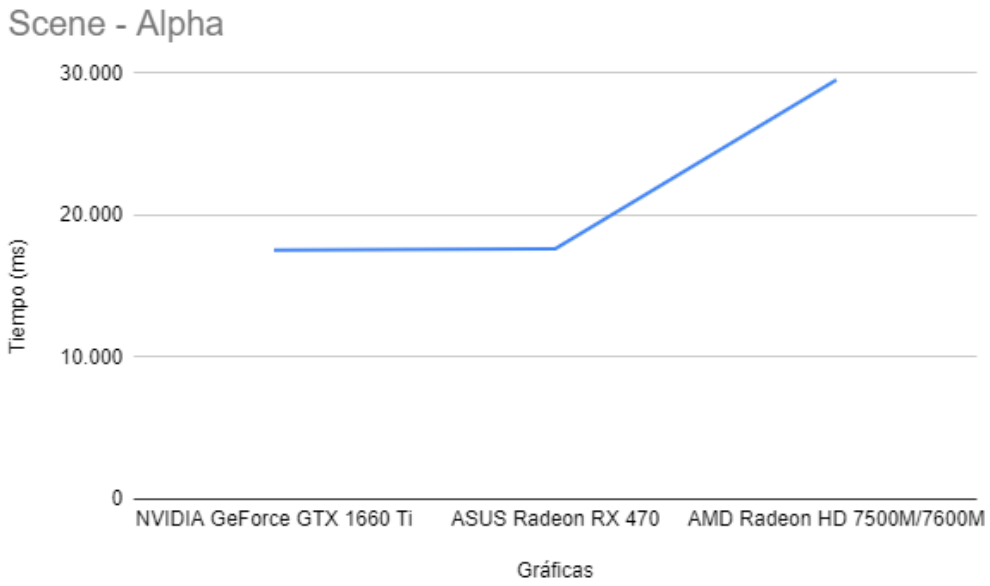

Figura 30 Gráfico Scene con técnica Alpha. Tiempo(ms) vs gráficas

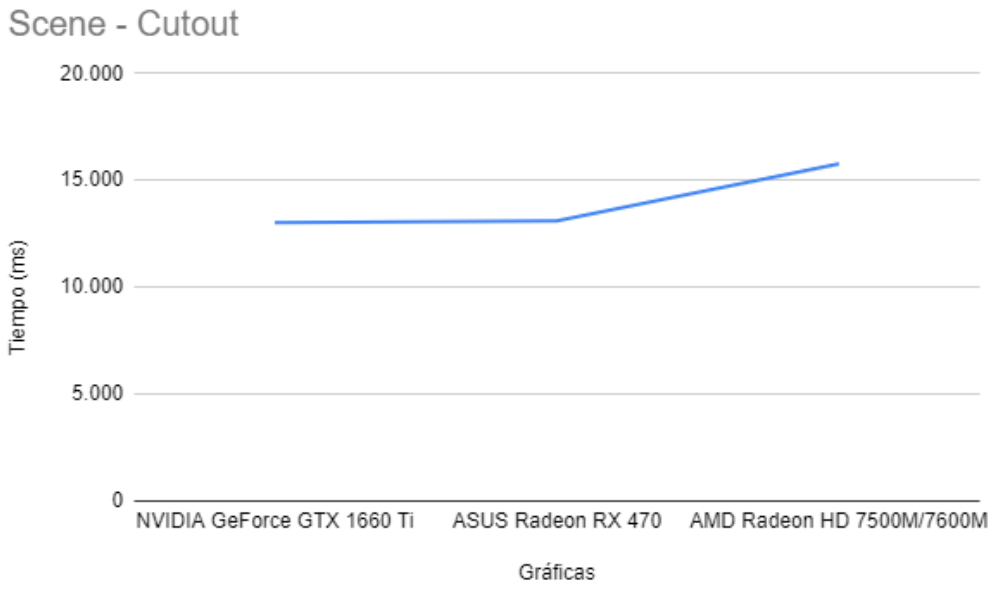

Figura 31 Gráfico Scene con técnica Cutout. Tiempo(ms) vs gráficas

Silvia Osoro García | silwia.o.g@gmail.com

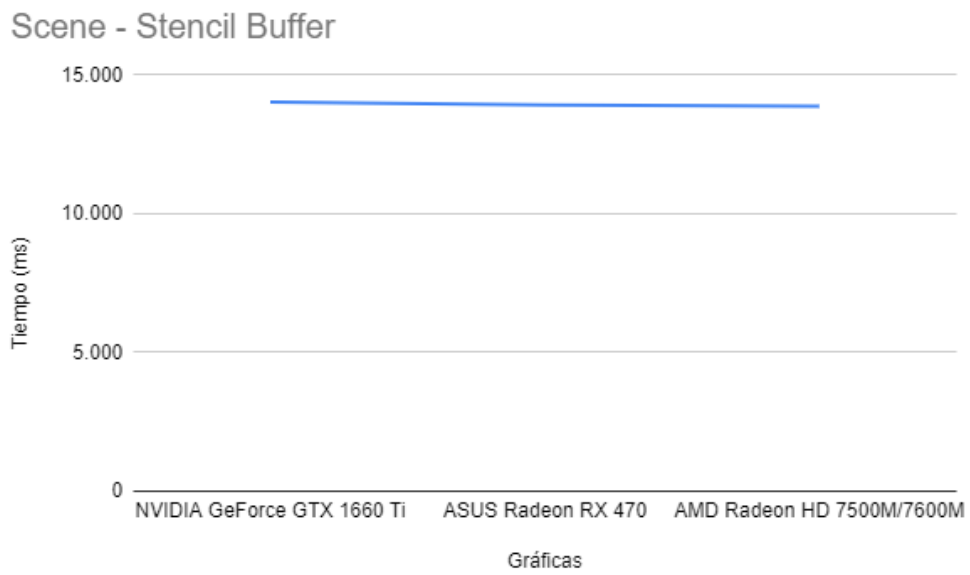

Figura 32 Gráfico Scene con técnica Stencil Buffer. Tiempo(ms) vs gráficas

Conclusiones

Podemos ver qué técnica es mejor en cada caso según: cuántos objetos en escena vamos a aplicarle la técnica, número de triángulos de los objetos al que le aplicamos la técnica, y capacidad de las gráficas de los dispositivos a donde dirigimos nuestro juego.

Vemos que el **Stencil Buffer**, tanto en FPS como en tiempos, es el **mejor** en cada una de las gráficas, ya sea por su **estabilidad**, **tiempos y soporte** entre cada gráfica y en cantidad de objetos a los que se les aplica y su cantidad de triángulos.
También vemos que hay situaciones en las cuales podemos hacernos la pregunta del principio de la investigación y elegir otra técnica: ¿cuánto rendimiento puedo permitirme **sacrificar**?, es ahí donde tenemos que seguir con: ¿Qué **características** tiene mi juego en común a estos datos? Objetos, tipo de objetos, capacidad de la gráfica del dispositivo o consola final… Con estas pruebas podemos hacernos una idea aproximada de **qué podemos usar en cada momento**.

A partir de aquí, podemos **balancear** con el tiempo de desarrollo para cada técnica y lo que requiere cada una a nivel de detalle hasta dejarla **visualmente** "mejor" o más interesante. Por ejemplo, el **Stencil Buffer** puede tener un acabado mucho más desarrollado con formas cambiantes como máscaras, imitando un noise con esas formas, usar partículas, etc.; las disoluciones con el **Cutout** suelen ser muy usadas por el buen acabado al que se puede llegar algo más rápido y por supuesto, con conocimiento en shaders; y el **Alpha** sería el más sencillo con un acabado bueno y rápido. Esta parte ya queda a necesidades y requisitos visuales de cada uno.

Por las características de mi juego, finalmente escogí el **Stencil Buffer**, ya que iría escalando la cantidad de objetos a los que se le aplicará (recordamos que también vamos a mostrar otros objetos a cambio de los que ocultamos) según sigamos desarrollando, y visualmente le podemos ir dedicando más tiempo hasta la finalización del proyecto. Hay que añadir que en la mecánica íbamos a sacrificar algo de rendimiento a cambio de que el cambio de objetos fuese el mínimo. Por otra parte, recordamos que las consolas finales en las que queremos publicar el juego necesitan que estén bien optimizados.

3. Metodología y organización

3.1 Trabajo en equipo

Un proyecto como este implica una buena previsión de riesgos, organización y gestión de los recursos con los que contamos. Para ello decidimos adoptar metodologías como: Scrum, por la flexibilidad, los sprints y el ritmo de reuniones; Kanban, que nos ha ayudado a dividir los contenidos de manera eficiente (checklist).

Al ser todos los miembros del proyecto estudiantes, el tiempo con el que contábamos para el desarrollo del juego era limitado, por lo que estos procesos nos han ayudado a paliar esto. A eso se suma el haberle dedicado tiempo a decidir el scope del juego para que estas herramientas sean efectivas sin necesidad de recortar funcionalidades del juego y reajustar la perspectiva sin que se diluyese demasiado.

La duración de los sprints era de dos semanas, y entre ellas, además de las reuniones para iniciar el sprint, hacíamos de una a dos reuniones de planificación y una sesión grupal de trabajo. Este tipo de reuniones nos ayudaba a saber cómo iba avanzando cada miembro del proyecto, con cuánto tiempo disponía y el esfuerzo que iba a requerir la tarea, y así poder adaptarlas.

Todo esto lo complementamos con el uso de distintas herramientas para organizarnos como son Trello (Organización mediante tablones), Discord (Comunicación entre integrantes del proyecto), GitHub (Repositorios donde se almacena la programación) y Google Drive (Almacenamiento de todo lo relacionado con el proyecto).

3.2 Trabajo personal

Ya que somos un equipo pequeño, cada uno tenía un departamento a su cargo, por lo que todo el departamento de programación caía sobre mí. Lógicamente, nos ayudamos entre los departamentos en aquello que podíamos. Las tareas que he realizado son:

3.2.1 Complementar al departamento de diseño y producción

Proponer y adaptar con diseño funcionalidades parecidas a las requeridas para que sean más viables a realizar con todo el conjunto de trabajo que hay.

3.2.2 Decisiones sobre herramientas

Decisiones de implementación basadas en la tecnología teniendo en cuenta las necesidades del diseño. Es decir, decidir si necesitábamos realmente cierta herramienta o assets para el proyecto (por espacio en el GitHub y en el proyecto en sí) o si podíamos implementarlo de otra manera, como, por ejemplo, la herramienta Digger, que es una herramienta para modificar el Terrain, ya que iba a ocupar un espacio considerable, vi mejor opción un proyecto a parte para

esa herramienta, y una vez finalizada la tarea, se llevase el Terrain terminado al proyecto.

3.2.3 Cambios de versión en Unity

Para mantener el proyecto actualizado en versiones estables o que solucionasen errores de las versiones en las que estábamos, cada tres meses aproximadamente íbamos revisando si venía bien subir el proyecto de versión.

3.2.4 Integración de herramientas al proyecto

- Yarn Spinner: esta herramienta es un motor de diálogos. Con la documentación que tienen en su página web lo logré implementar y poner en funcionamiento junto con unos ejemplos de uso para posteriormente a partir de ellos hacer los diálogos que necesitemos. A parte hice un documento con los prefabs que había que usar en el proyecto y los scripts que llevaba cada uno para que funcionase la herramienta.

- Incontrol: esta herramienta es un administrador de inputs (entradas que recogen las acciones del jugador) nos proporcionaba un mapeado estandarizado, facilitándonos el pasar de una plataforma a otra, por ejemplo, tendremos el teclado, el mando de PlayStation y los controles de Switch añadiendo unas pocas líneas.

- Naughty Attributes: es una extensión de los atributos de Unity para hacer que podamos tener un inspector más potente sin la necesidad de usar editores personalizados.

- Packs: hemos implementado algún que otro pack de assets, pero el más importante es el mencionado anteriormente, The Illustrated Nature Pack, el cual proporcionaba shaders como nieve, o herramientas para cambiar el color de los objetos, el cual usé a mi favor para el cambio de estación, entendiendo sus scripts y modificándolos.

- FMOD lo integró y usó exclusivamente Jorge Aranda López.

- Terrain Tools y Prefab Painter lo integró y usó exclusivamente Jorge Aranda López

3.2.5 Movimiento del personaje

Desarrollé el movimiento completo del personaje: correr, volar, nadar y bucear.
Desde Unity, tuve que elegir cómo mover al personaje, si por físicas con el Rigidbody o con el Player Controller, y finalmente decidí por el Player Controller principalmente porque se adapta mejor a cualquier tipo de suelo con rugosidades, cuestas, etc., y ya que el movimiento es en una isla, esto era esencial.

3.2.6 Creado el sistema de misiones.

El sistema se compone de un script manager que controla qué pasa si cogemos una misión (las demás tienen otro diálogo), qué pasa si terminamos esa misión. Tiene constancia de todas las misiones y sus diálogos en cada momento.

3.2.7 Creado el cambio de estación

Para el cambio de estación, cogí de base los scripts hechos en el pack mencionado anteriormente en el que se recogía los colores de cada material. Después de estudiar lo que hacía y desde dónde, lo modifiqué para que cambiase de colores mezclando el color actual con el siguiente que hayamos elegido.

Aplicando la investigación, empleé la técnica del stencil buffer para que aparezcan objetos únicos de la nueva estación y desaparezcan aquellos que son de la anterior, cambiándoles de layers e instanciando esferas que hacen de máscara para mostrar u ocultar los objetos.

3.2.8 Creado e implementado el menú e interfaz

Creé un menú de inicio para jugar o salir del juego y opciones para cambiar el brillo, la calidad de imagen, el tamaño de la ventana y configurar el audio. Podemos navegar por el mismo con mando o teclado. Este menú guarda los cambios en Player Prefs y se aplican a la escena de juego cuando entramos o si volvemos a entrar en el juego.

También implementé que pudiéramos navegar por la interfaz de diálogo con el teclado y mando, aparte de poder seleccionar e interactuar con el ratón.

3.2.9 Control de versiones

Encargada de aprender y enseñar al equipo el correcto funcionamiento de GitHub, las versiones del proyecto, y arreglar los problemas que surjan.

Hice un documento con los conocimientos y pasos básicos en GitHub Desktop para que el equipo pudiese consultar en caso de tener dudas más básicas. Está en Anexos > Anexo_Programacion_OtraDocumentacion.

4. Desarrollo

En este apartado dejaré constancia del trabajo realizado por mí, más concretamente, para el TFG y los antecedentes a la realización de este.

4.1 Perfil

Durante los últimos 3 años de carrera he centrado mi perfil como programadora de gameplay y en ser capaz tanto de llevar proyectos en solitario en el departamento de programación, como trabajar en equipo en el mismo.
Esta es la primera vez que llevo el departamento de programación entero, sola, en un juego tan grande como este.
Mi objetivo era ser capaz de encargarme de cada parte del desarrollo dentro del departamento de programación para aprender mucho más y acercarme a ser una programadora polivalente, en el sentido de que, aunque me especialice, sepa hacer de todo dentro de mi ámbito de programación. Aparte, empezar a adentrarme al mundo de la optimización, procesos de la Render Pipeline y estudio de datos me ha ayudado a querer empezar a especializarme en este ámbito.

4.2 Tecnología utilizada

El proyecto empezó en la versión 2019.3.0f5, y actualmente usa la versión 2020.3.7f1 de Unity. Por otra parte, he usado las herramientas mencionadas anteriormente, GitHub como repositorio, Visual Studio 2019 y Trello.

4.3 Trabajo realizado

La creación del juego comenzó a mediados del año 2020. Voy a explicar mi trabajo realizado como programadora en el proyecto desde entonces.

Antes del desarrollo vinieron dudas importantes, la que más, fue si usar la URP de Unity o no. Por inexperiencia con las Render Pipelines en general y la poca documentación y exploración sobre la URP respecto a la estándar de Unity, ponían un gran peso en la balanza para continuar el proyecto en la estándar. Aunque fuese más costoso, finalmente decidí la URP por los siguientes motivos: es una oportunidad de empezar a conocer más sobre esa pipeline; iba a ayudar a que el juego se viese mejor con menos recursos y menos esfuerzo por parte de los desarrolladores; y la facilidad de cambiar entre calidades de imagen hicieron que la URP fuese la elegida. Si bien es cierto que propuso unos retos distintos y tal vez menos información, como por ejemplo, a la hora de usar el Stencil Buffer o reutilizar shaders ya no compatibles con la pipeline, el enfrentarme a un reto así a la vez que desarrollaba todo el resto de la programación del proyecto me hizo crecer enormemente como profesional.

Empecé desarrollando el movimiento y la cámara del personaje. Probé el movimiento a través de Rigidbodies y Character Controller. Tras analizar los pros y los contras, decidí usar Character Controller para el movimiento, puesto que me facilitaba el movimiento en sitios que no fuesen planos, y ese punto era muy importante para que el personaje se moviese sin problemas por la isla. Empecé a plantear cada movimiento y a plasmarlo.

Respecto a la cámara, en un principio la programé con movimiento libre y detectando colisiones.

Finalmente, el movimiento ha quedado con un movimiento por ejes, los cuales se mantienen con la posición de la cámara, es decir, si pulsamos la tecla D, siempre iremos hacia la derecha en perspectiva de la cámara.
El planeo se activa tras unos pocos segundos que mantengas pulsado la tecla correspondiente y cambia todos los valores que tiene el movimiento, además de cómo actúa la gravedad, simulando que no está realmente cayendo si no poco a poco bajando.

Figura 33 Vuelo

El nado se activa calculando la distancia que hay entre cierto objeto de referencia (hijo del player) y la superficie del agua, que es un box collider. Nos movemos entre límites. No tendremos gravedad. La variable de movimiento será de impulso: se acelera hasta cierto límite, y rápidamente la variable baja hacia el otro límite.
Para el buceo, teniendo en cuenta los límites del nado (no sobrepasar la superficie) desplazamos en el eje Y al personaje.

Figura 34 Personaje Nadando

La cámara usa la herramienta Cinemachine. Al jugador le siguen varias cámaras virtuales, las cuales vamos activando y desactivando para que nuestra cámara tome cierta posición de una cámara virtual u otra. En el mapa tenemos zonas marcadas por triggers en las que entras y se activa una de esas cámaras. Es importante dejar una estándar o main.

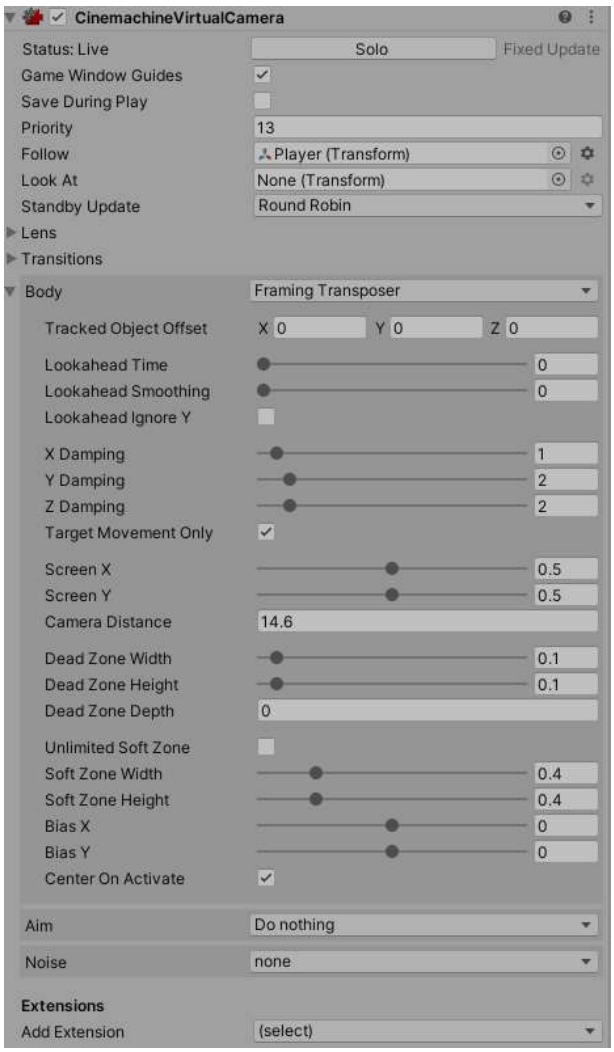

Figura 35 Configuración cámara virtual Cinemachine

Otra tarea importante fue el sistema de misiones. Cada nota es consciente de cuál es su objeto para enviarle cuándo ha sido recogida la nota, y por tanto, que cambie su diálogo y se prepare para ser recogido por el jugador una vez lo encuentre.

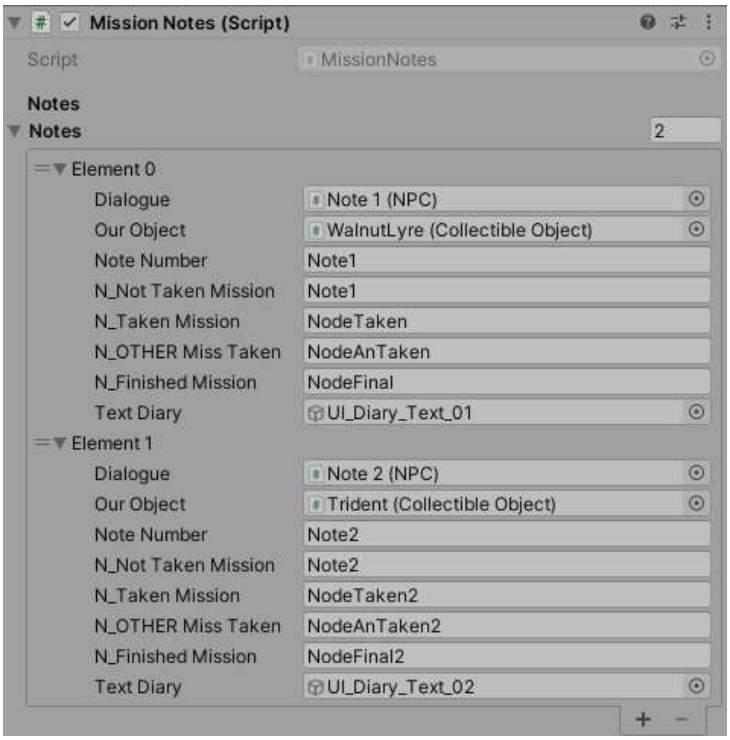

Figura 36 Notas de misión

Cada vez que hay un cambio, las demás notas también cambian sus diálogos. Todo está preparado para que el diseñador inserte los nodos de diálogo que se quiere ejecutar en cada situación (si ya se cogió esa nota, si ya cogimos su objeto, si estamos en otra misión, etc).

Figura 37 Diálogo 1

Una vez tenemos todas las misiones hechas y por ende, todos los objetos divinos recogidos, comienza el cambio de estación.

Antes de hablar del cambio de estación, mencionar que en el pack de IL3DN proporcionaba un par de Managers con los colores de ciertos materiales asignados por cada slot, un total de 4 (uno por cada estación). Con ello de base, antes de hacer el cambio, guardo el anterior color y creo una corrutina para que con ese anterior color y el siguiente al que queremos cambiar, hacemos un blend.

```
private IEnumerator SetColorGradually()
{
    Color[,] colors = new Color[materials.Count, 5];
    for (int i = 0; i < materials.Count; i++)
    {
        for (int k = 0; k < materials[i].properties[materials[i].previousProperty].colors.Count; k++)
        {
            colors[i, k] = (materials[i].properties[materials[i].previousProperty].colors[k].color);
        }
    }

    for (float t = 0.0f; t < secondsChange / 10; t += Time.deltaTime)
    {
        for (int i = 0; i < materials.Count; i++)
        {
            for (int k = 0; k < materials[i].properties[materials[i].selectedProperty].colors.Count; k++)
            {
                string propertyName = materials[i].properties[materials[i].selectedProperty].colors[k].name;
                materials[i].meterial.SetColor(propertyName,
                    Color.Lerp(colors[i, k], materials[i].properties[materials[i].selectedProperty].colors[k].color, t));
            }
        }
        yield return null;
    }
}
```

Figura 38 Mi añadido en la herramienta IL3DN

Con esto, cuando comienza el cambio de estación, llamamos a esta corrutina indicándole cuál es la siguiente estación. El scope inicial es de verano a invierno, y como quise dejarlo preparado para las demás estaciones, de momento avanza un contador saltándose un número cada vez que cambia (del 1 al 3 y del 3 al 1). Vemos si la siguiente estación va a ser invierno, y activamos la herramienta de nieve y vamos aumentando sus valores. Aquí vamos a aplicar lo visto en la **investigación**, y es que cada objeto que vamos a cambiar por la estación, se le crea un objeto máscara (en mi caso esfera). A estos objetos, al principio del cambio, les cambiamos de layer a “SeeThrough” y hacemos que vayan creciendo (escalando) las máscaras, de manera que las van ocultando, y los objetos que queremos que vayan apareciendo, las ponemos en la layer “ReverseMask”. Una vez finalizado el cambio, volvemos a poner la escala de la máscara a 0 e intercambiamos sus layers, es decir, la que estaba en “SeeThrough” por la “ReverseMask” y viceversa.
La layer de SeeThrough al pasar por la máscara del stencil no se dibujará, y con ReverseMask al pasar por la máscara se irá dibujando el objeto en resumidas cuentas.

Figura 39 Cambio de estación a invierno

Otra de las tareas fue implementar la herramienta de textos Yarn Spinner. Mediante la documentación aportada en su página web, la fui implementando, estudiando los ejemplos y haciendo pruebas para explorar cada funcionalidad de la herramienta, hice ejemplos simulando el objetivo por parte de diseño: una misión y una nota para dejar una plantilla que solo necesite de cambiar lo que dicen los textos.

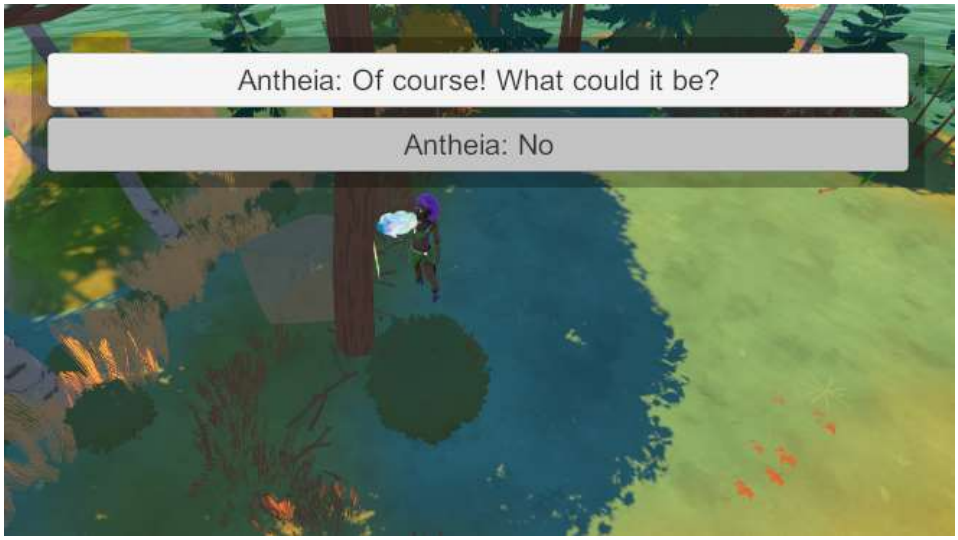

Figura 40 Diálogo 2

El InControl ayudó mucho a la hora de mapear los inputs. Siguiendo la documentación aportada en la página web y los ejemplos que tienen en Unity hice mi plantilla con los inputs que necesitaba y con la facilidad de añadir cuantos quisiese.

```
public MyPlayerActions()
{
    Fire = CreatePlayerAction( "Fire" );
    JumpGlide = CreatePlayerAction( "JumpGlide" );
    Left = CreatePlayerAction( "Move Left" );
    Right = CreatePlayerAction( "Move Right" );
    Up = CreatePlayerAction( "Move Up" );
    Down = CreatePlayerAction( "Move Down" );
    DiveUp = CreatePlayerAction( "DiveUp" );
    DiveDown = CreatePlayerAction( "DiveDown" );
    Move = CreateTwoAxisPlayerAction( Left, Right, Down, Up );
    Interacion = CreatePlayerAction("Interacion");
    Exit = CreatePlayerAction("Exit");
    Diary = CreatePlayerAction("Diary");
}
```

Figura 41 Incontrol: Creación de los bindings

También, la UI fue un buen momento para usar PlayerPrefs y experimentar con lo que te proporciona Unity, ya sea para el cambio de quality settings, modificar y personalizarlos, o acceder a las resoluciones de la pantalla y mostrarlas. Con la documentación de Unity se hizo muy accesible crear el menú. Además, configuré que se pudiese navegar por la UI con el teclado y mando, incluso en las conversaciones.

Figura 42 Menú de opciones

Además del proyecto, el camino por GitHub tuvo sus complicaciones, sobre todo por falta de experiencia. Una vez entendido cómo es que el repositorio estaba almacenando basura y cómo solucionarlo, fue más sencillo trabajar con GitHub sin que saliesen más errores.

Añadir que intenté aprender cómo hacer un control de versiones de manera un poco más profesional y me pareció un sistema mucho más eficaz. Teníamos dos branch: master y develop, en la master, cada vez que había una versión estable, mergeaba ambas branches. Todo el desarrollo realizado por mi y mis compañeros está en la branch de develop, posteriormente mergeada. Esto nos facilitaba encontrar fallos que no habíamos identificado en el momento en el que subimos cierto cambio y nos aseguraba tener una versión estable cubriéndonos las espaldas. Utilizamos números de versión para siempre asegurarnos que estamos en el número correcto (avisar que lo último subido es en la 0.8.7, por ejemplo) y así obligarnos a volver a mirar el siguiente número de versión que tenemos que subir en History (cosa que podremos ver si estamos realmente en la última) y de esta manera evitar que no nos bajemos los cambios anteriores antes de, por lo menos, subir nuestros cambios.

Figura 43 Uso del GitHub

También hice un documento para dejar constancia del uso de GitHub en Anexos / OtraDocumentacion / Nociones básicas de GitHub.pdf

Por último, dejé hechos prefabs para ayudar a montar los escenarios, a parte de documentación tanto para el equipo como para mí para ayudar a montarlos con las herramientas que tenemos y lo que he ido haciendo para que el proyecto funcione. Se pueden ver en Anexos / OtraDocumentacion / Implementación del proyecto.pdf, y Sistema de misiones.pdf.

5. Resultados

Los resultados del proyecto final de grado A Seed's Tale después de haberlo realizado en su totalidad, han sido muy satisfactorios. Se han cumplido la mayoría de los objetivos marcados en su inicio y esperamos una gran acogida por el público en el momento de su lanzamiento en la plataforma de PC. La fecha prevista de lanzamiento que nos hemos marcado es a finales de 2021, previa búsqueda de un publisher que nos ayude en el momento de lanzamiento con financiación.
Dicha financiación nos ayudará a terminar los últimos detalles del juego que no se han terminado de pulir durante el desarrollo del proyecto, por ejemplo, el resto de las estaciones con sus misiones, implementar el patinaje, terminar el vuelo, objetos mortales y de decoración, pantalla de carga, animaciones finales, y hacer el port a Switch.
También está previsto el lanzamiento de merchandising que acompañará al lanzamiento del título, chapas, pegatinas, libro de arte con ilustraciones y bocetos de los escenarios de cada estación y banda sonora.

6. Conclusiones del proyecto

Tras lo dicho anteriormente, se puede concluir que A Seed's Tale ha sido un éxito, tanto a nivel académico como personal. El aspecto económico lo podremos ver después del lanzamiento. Ha sido un trayecto de un año entero de trabajo en el que considero que he mejorado mucho como programadora, aplicando y mejorando técnicas vistas en la carrera y externas, nuevos conceptos y experiencia en el terreno 3D y de desarrollo de videojuegos en general.

A Seed's Tale comenzó como un proyecto asequible para un grupo pequeño, con ambición de poder aprender en el camino del desarrollo aquello que tuviésemos pasión o simplemente curiosidad por investigar y llevar a cabo en este último año de carrera. Las ideas más claras que teníamos en mente era hacer un juego con la mecánica del cambio de estación como principal, que fuese en un entorno natural y que, sobre todo, fuese totalmente viable y tuviésemos motivación por seguir desarrollando el juego durante un año tan duro.
Desde esas ideas iniciales, estudiando sobre mitología griega, empezamos a conectar, formar una historia y definir el juego.

El juego se ha mantenido con el objetivo y tareas planificadas, solo iban cambiando según íbamos obteniendo recursos que nos permitiesen avanzar o pulir lo ya hecho, por ejemplo, gracias al pack de naturaleza no nos tuvimos que preocupar por los shaders, y facilitó el cambio de estación, lo que nos permitió seguir puliendo y rehaciendo partes del movimiento para mejorarlo.

Decidimos muy bien qué iba a tener y qué no iba a tener el juego según nuestras capacidades y previendo a dónde podíamos llegar desarrollando partes que no habíamos hecho nunca. Esto nos ayudó mucho a medir de manera rápida los problemas que pueden surgir y tener previsto tiempo para solventarlos. A parte de que ayuda mucho a conocer nuestros límites y así jugar con ellos o incluso expandirlos.

En nuestro caso, el análisis de las distintas tecnologías y nuestra experiencia previa con Unity, hicieron que el proceso de creación del juego no tuviese tantos obstáculos, permitiéndonos centrarnos en el desarrollo del juego sin esos problemas añadidos, pudiendo así dedicar más tiempo al juego y a los detalles.

7. Webgrafía

- G. Blázquez, D. (2006). Optimización de gráficos 3D para videojuegos. 02/12/2020, de ExeBlog Sitio web: http://www.exelweiss.com/blog/18/optimizacion-de-graficos-3d-para-videojuegos/#:~:text=En%20el%20mundo%20de%20los,%2C%20los%20gráficos%2C%20los%20sonidos…

- Dunn, A. (2014). Transparency (or Translucency) Rendering. 02/12/2020, de NVIDIA Developer Sitio web: https://developer.nvidia.com/content/transparency-or-translucency-rendering

- Tokio School. (s.f). ¿Qué son las texturas para videojuegos? 04/12/2020, de Tokio New Technology School Sitio web: https://www.tokioschool.com/noticias/texturas-para-videojuegos/

- Denham, T. (s.f). What is 3D & Game Shaders? 06/12/2020, de Concept Art Empire Sitio web: https://conceptartempire.com/shaders/

- Shehata, O. (2015). A Beginner's Guide to Coding Graphics Shaders. 06/12/2020, de Envato Tuts+ Sitio web: https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-coding-graphics-shaders--cms-23313

- White, S., Coulter, D., Batchelor, D., Satran, M.. (2018). Stencil Buffer Techniques (Direct3D 9). 03/02/2021, de Microsoft Docs Sitio web: https://docs.microsoft.com/en-us/windows/win32/direct3d9/stencil-buffer-techniques?redirectedfrom=MSDN

- White, S., Coulter, D., Batchelor, D., Satran, M., Jacobs, M. (2018). Graphics pipeline. 07/12/2020, de Microsoft Docs Sitio web: https://docs.microsoft.com/en-us/windows/win32/direct3d11/overviews-direct3d-11-graphics-pipeline?redirectedfrom=MSDN

- Lawrence, J., Kazhdan, M., Klein, A., Funkhouser, T., Finkelstein, A., Dobkin, D. (2012) 3D Polygon Rendering Pipeline. 07/12/2020, de web.archive.org Sitio web: https://web.archive.org/web/20161229102150/http://www.cs.virginia.edu/~gfx/Courses/2012/IntroGraphics/lectures/13-Pipeline.pdf

- Sikachev, P. (s.f). Depth proxy transparency rendering. 08/12/2020, de Eidos Montreal Sitio web: https://www.eidosmontreal.com/news/depth-proxy-transparency-rendering/

- Arm Developer. (s.f). Material and Shader Best Practices for Artists. 08/12/2020, de Arm Developer Sitio web: https://developer.arm.com/solutions/graphics-and-gaming/developer-guides/game-artist-guides/material-and-shader-best-practices/single-page

- Owens, B. (2013). Forward Rendering vs. Deferred Rendering. 08/12/2020, de Envato Tuts+ Sitio web: https://gamedevelopment.tutsplus.com/articles/forward-rendering-vs-deferred-rendering--gamedev-12342

- Lammers, K. (2013). Unity shaders and effects cookbook: Discover how to make your Unity projects look stunning with shaders (Pdf ed.). Packt Publishing Ltd. pp. 143-151, 173-185

- OpenGL (s.f). Dissolves with Stencil. 03/02/2021, de OpenGL Sitio web: https://www.opengl.org/archives/resources/code/samples/advanced/advanced97/notes/node197.html#:~:text=The%20stencil%20buffer%20can%20be,%2C%20smoothing%2C%20and%20other%20effects

- University of Helsinki. (s. f.). z-Buffer Algorithm. cs.helsinki.fi. Recuperado 3 de febrero de 2021, de https://www.cs.helsinki.fi/group/goa/render/piilopinnat/z_buffer.html

- Shahrizal Sunar, M., & Kolivand, H. (2011). To Combine Silhouette Detection and Stencil Buffer for Generating Real-Time Shadow (1.a ed., Vol. 2). International Journal of Computer Graphics.

- Technologies, U. (s. f.). Unity - Manual: Transparent Cutout Shader Family. Unity Documentation. Recuperado 3 de febrero de 2021, de https://docs.unity3d.com/Manual/shader-TransparentCutoutFamily.html

8. Anexos

8.1 Anexo de Programación

1. Documentación.

1.1. Análisis.

Requisitos técnicos según Unity para la build:

Operating system	Windows	Universal Windows Platform	macOS	Linux
Operating system version	Windows 7 (SP1+) and Windows 10	Windows 10, Xbox One, HoloLens	High Sierra 10.13+	Ubuntu 20.04, Ubuntu 18.04, and CentOS 7
CPU	x86, x64 architecture with SSE2 instruction set support.	x86, x64 architecture with SSE2 instruction set support, ARM, ARM64.	x64 architecture with SSE2.	x64 architecture with SSE2 instruction set support.
Graphics API	DX10, DX11, DX12 capable.	DX10, DX11, DX12 capable GPUs.	Metal capable Intel and AMD GPUs	OpenGL 3.2+, Vulkan capable.
Additional requirements	Hardware vendor officially supported drivers. For development: IL2CPP scripting backend requires Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.	Hardware vendor officially supported drivers. For development: Windows 10 (64-bit), Visual Studio 2015 with C++ Tools component or later and Windows 10 SDK.	Apple officially supported drivers. For development: IL2CPP scripting backend requires Xcode. Targeting Apple Silicon with IL2CPP scripting backend requires macOS Catalina 10.15.4 and Xcode 12.2 or newer.	Gnome desktop environment running on top of X11 windowing system Other configuration and user environment as provided stock with the supported distribution (such as Kernel or Compositor) Nvidia and AMD GPUs using Nvidia official proprietary graphics driver or AMD Mesa graphics driver.

Figura 44 Requisitos build Unity

Plataforma: PC, soportado en Windows 7/8/10, que es nuestro objetivo de salida en un principio.
Tipos de gráficos: 3D no procedurales por el estilo de juego de aventura en una isla.
Periféricos de entrada: Teclado y ratón por ser en PC, y mando para aquellos jugadores que quieran, puedan elegir.
Modos de juego: Individual porque el objetivo eres tú y tu entorno dentro del juego.
Herramientas de desarrollo: Las explico más detalladas en el **2.1 Decisiones Iniciales**

1.2. Diagrama de componentes.

Ver en Anexos / Anexo_Programacion_DiagramaDeComponentes

1.3. Diagrama de clases.

Ver en Anexos / Anexo_Programacion_DiagramaDeClases.

1.4. Librerías de terceros.

He usado Incontrol, Yarn Spinner y Naughty Attributes, a parte del pack de IL3DN. FMOD, Prefab Painter y la herramienta Terrain de Unity no las he implementado ni usado. Las explico más detalladas en el **2.1 Decisiones Iniciales y** **3.2.4 Integración de herramientas al proyecto**

1.5. Roles de programación.

He desarrollado toda la programación del proyecto (excepto la que viene de librerías externas y la implementación de audio) en A Seed's Tale. Hay información mucho más detallada y extensa en dos puntos en los que ya hablo sobre mi rol: **3.2 Trabajo personal** y **4.3 Trabajo realizado**.

1.6. Documentación del código.

Ver en Anexos / Anexo_Programacion_Documentación.

1.7. Manual.

El proyecto está destinado a la plataforma PC en Windows principalmente.
No requiere instalación previa, se ejecuta mediante un archivo .exe.
Admite teclado y ratón, y mando.
No se requiere de conexión a Internet para disfrutar de toda la funcionalidad del software.

1.8. Otra documentación.

Ver en Anexos / Anexo_Programacion_OtraDocumentación.

2. Proyecto

2.1. Requisitos para usar el Proyecto.

Se necesitará una versión de Unity 3D para poder generar el proyecto que contiene el juego A Seed's Tale. Se recomienda que sea la versión 2020.3.7f1 que es la versión actual del proyecto, para evitar errores de conversión entre versiones del motor.
Si se quiere probar el proyecto desde Unity, para no generar errores de ejecución, se deberá lanzar desde la escena MainMenu.

2.2. Estructura del Proyecto.

Estructura:

|--- Editor Default Resources
(Esta carpeta la genera FMOD, y si la cambio de sitio daría error)
|--- Gizmos
(Esta carpeta la genera FMOD, y si la cambio de sitio daría error)
|--- IL3DN
(Esta carpeta es de un pack de Unity. La dejo fuera porque es una herramienta importante y que los desarrolladores mantienen activa, por lo que, si lo quiero actualizar, necesito dejarlo fuera)
|--- InControl
(Esta carpeta es de un pack de Unity. La dejo fuera por los mismos motivos que IL3DN)
|--- MyProject
| |--- Animations (Las animaciones y animators del proyecto)
| |--- Audio (Todos los audios del proyecto)
| |--- DialogueYS (Los diálogos que se van a usar en el juego (de YarnSpinner))
| |--- Materials (Materiales, Texturas y Sprites)
| |--- Models (Modelos y objetos del proyecto)
| |--- Prefabs (Nuestros prefabs)
| |--- Scenes (Las escenas del proyecto)
| |--- Scripts (Mis scripts)
| | |--- Scripts Externos (Scripts que no he hecho yo)
| |--- TechnicalArt (Aquí están shaders generados por nosotros y Postprocesados)
| |--- Terrain (Terrain, brushes, etc. del Terrain Tool)
| |--- Video (El vídeo de la intro)
|--- nTools
(Esta carpeta es de un pack del Prefab Painter, y si lo actualizasen, Unity lo buscaría en Assets y lo actualizaría ahí.)
|--- Plugins
(Esta carpeta es propia de Unity, y las herramientas que no iba a actualizar las he metido ahí, excepto FMOD que al instalarlo se metió ella sola)
|--- StreamingAssets
(Esta carpeta es generada por Unity)
|--- URP
(Esta carpeta es propia de Unity y desde ahí es muy accesible)

2.3. Fuentes del Proyecto.

Ver en Anexos / Anexo_Programación_Fuentes.

3. Binarios

3.1. Binario distribuible

Ver en Anexos / Anexo_Programación_Binario.

8.2. Estructura de los contenidos

Memoria_Invidual_ASeedsTale_SilviaOsoroGarcia
Memoria_Individual_ASeedsTale_SilviaOsoroGarcia.pdf
Anexos
o Anexo_Investigacion_Binario
BUILD_ALPHA
Object
Alpha_Object.exe
Scene
Alpha_Scene.exe
BUILD_CUTOUT
Object
Cutout_Object.exe
Scene
Cutout_Scene.exe
BUILD_STENCIL
Object
Stencil_Object.exe
Scene
Stencil_Scene.exe

o Anexo_Investigacion_Fuentes
SilviaTest.rar

o Anexo_Programacion_Binario
ASeedsTale.exe

o Anexo_Programacion_DiagramaDeClases
ClassDiagram.png

o Anexo_Programacion_DiagramaDeComponentes
ComponentDiagram.png

o Anexo_Programacion_Documentacion
Documentacion.rar

o Anexo_Programacion_OtraDocumentacion
Nociones_básicas_de_GitHub.pdf
Implementación_del_proyecto.pdf
Sistema_de_misiones.pdf

o Anexo_Programacion_Fuentes
ASeedsTale.rar

Silvia Osoro García | silwia.o.g@gmail.com