UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Politécnica Superior de Gandia

Producción de efectos visuales 3D a partir de sistemas de partículas y fluidos.

Trabajo Fin de Máster

Máster Universitario en Postproducción Digital

AUTOR/A: Siles Merino, Pau

Tutor/a: García Miragall, Carlos Manuel

CURSO ACADÉMICO: 2022/2023

Resumen

En este trabajo de final de Máster se presenta el proceso de producción de diferentes efectos visuales 3D, los cuales están generados mediante sistemas de partículas y fluidos en tiempo real. Para ello se ha efectuado una contextualización de la historia y la evolución de los efectos visuales, y se ha recopilado información sobre los sistemas de partículas y fluidos. También se han explicado diferentes fundamentos básicos del renderizado en tiempo real y se han determinado los principales *softwares* que se utilizan para ello. Tras diseñar un flujo de trabajo, se han generado diferentes cinemáticas que contienen efectos visuales y se ha explicado su proceso de creación, permitiéndome aprender los diferentes procesos y problemáticas que engloban la generación de efectos visuales.

Palabras Clave

Efectos visuales; VFX; 3D; Tiempo real; Unreal Engine

Abstract

This Master's final project presents the production process of different 3D visual effects, which are generated by particle and fluid systems in real-time. To do this, first of all, a contextualization of the history and evolution of visual effects has been carried out, and information on particle and fluid systems has been compiled. Different basic fundamentals of real-time rendering have also been explained and the main software used for it has been determined. After designing a workflow, different cinematics with visual effects have been generated and their creation process has been explained, allowing me to learn the different processes and problems of the generation of visual effects.

Keywords

Visual effects; VFX; 3D; Real-time; Unreal Engine

Índice

1.- Introducción 5
1.1.- Motivación 5
1.2.- Objetivos 5
1.3.- Metodología 6
1.4.- Estructura de la memoria 7
2.- Marco Teórico / Contexto 8
2.1.- Efectos visuales. Definición y conceptos clave 8
2.2.- Historia y evolución de los efectos visuales 10
2.3.- Contextualización de los sistemas de partículas. Simulaciones de fluidos 15
2.4.- Renderizado en tiempo real 19
3.- Realización de casos prácticos 25
3.1.- Motores en tiempo real y aplicaciones 3D 25
3.1.1.- Unity 25
3.1.2.- Unreal Engine 27
3.1.3.- Cryengine 29
3.1.4.- Justificación del motor utilizado 30
3.1.5.- Otras aplicaciones 3D necesarias 30
3.2.- Pipeline 32
3.3.- Proyecto 1. Nieve y niebla 33
3.3.1.- Set 33
3.3.2.- Efectos visuales 35
3.3.3.- Render 38
3.3.4.- Composición y corrección de color 39
3.4.- Proyecto 2. Simulación de humo 40
3.4.1.- Set 40
3.4.2.- Efectos visuales 41
3.4.3.- Render 45
3.4.4.- Composición y corrección de color 46
3.5.- Proyecto 3. Simulación de fluidos 48
3.5.1.- Set 48
3.5.2.- Efectos visuales 49
3.5.3.- Render 50
3.5.4.- Composición y corrección de color 51
4.- Conclusiones 52
5.- Referencias Bibliográficas 53
6.- Anexos 55
Anexo 1. Especificaciones del ordenador utilizado 55
Anexo 2. Proceso de creación del set del proyecto 1. 55
Anexo 3. Proceso de creación del set del proyecto 2 57
Anexo 4. Proceso de creación del set del proyecto 3 58
Anexo 5. Videos de los proyectos. 60

Índice de tablas y figuras

Figura 1. Secuencia de créditos de Vertigo y computadora mecánica. 11
Figura 2. Primer uso de un sistema de partículas en Star Trek II 13
Figura 3. Efecto morphing en Terminator II 14
Figura 5. Sistema de fluidos generado con partículas en Houdini 17
Figura 6. Sistema de fluidos con partículas en Unreal Engine 18
Figura 7. Set de produción virtual para The Mandalorian 20
Figura 8. Material PBR en Unreal Engine 24
Figura 9. Interfaz de Unity. 26
Figura 10. Unreal Engine interface 29
Figura 11. Blender interface 31
Figura 12. Flujo de trabajo utilizado en este TFM. 32
Figura 13. Frame del resultado final del proyecto 1 33
Figura 14. Configuración de las olas 34
Figura 15. Océano creado para el proyecto 1 34
Figura 16. Set y vista *Nanite* de la geometría. 35
Figura 17. Sistema de partículas básico en Niagara. 35
Figura 18. Sistema de partículas en el set 36
Figura 19. Sistema de partículas emitiendo una textura 37
Figura 20. Blueprint con los sistemas de partículas y el spline 38
Figura 21. Render final proyecto 1 38
Figura 22. Corrección de color del proyecto 2 y comparación. 39
Figura 23. Frame del resultado final del proyecto 2. 40
Figura 24. Set e iluminación del proyecto 2 40
Figura 25. Sistema de partículas para el propulsor 41
Figura 26. Blueprint con el sistema de partículas 42
Figura 27. Creación de humo en Niagara. 43
Figura 28. Creación de la explosión. 44
Figura 29. Modo fractura en Unreal Engine. 44
Figura 30. Destrucción de la nave. 45
Figura 31. Frame del proyecto 2 46
Figura 32. Frame del proyecto 2 46
Figura 33. Corrección de color del proyecto 2 47
Figura 34. Comparación de la corrección de color del proyecto 2 47
Figura 35. Set del proyecto 3 48
Figura 36. Frame del resultado final del proyecto 3. 48
Figura 37. Set y vista *Nanite* del proyecto 3 49
Figura 38. Sistema de líquidos en Niagara. 50
Figura 39. Animación de las cámaras del proyecto 3. 50
Figura 40. Corrección de color del proyecto 3 51

Figura 41. Comparación de la corrección de color del proyecto 3.....51
Tabla 1. Características del equipo utilizado.....55
Figura 42. Set del proyecto 1 sin iluminación. 55
Figura 43. Set del proyecto 1 con iluminación. 56
Figura 45. Vistas Nanite de la geometría del proyecto 1. 56
Figura 46. Set del proyecto 2 sin iluminación. 57
Figura 47. Set del proyecto 2 con iluminación. 57
Figura 48. Vistas Nanite del proyecto 2. 58
Figura 49. Set del proyecto 3 sin iluminación. 58
Figura 50. Set del proyecto 3 con iluminación. 59
Figura 51. Vistas Nanite del proyecto 3. 59

1.- Introducción

1.1.- Motivación

El uso del 3D en la industria audiovisual ha revolucionado y cambiado la forma de realizar series, películas, y videojuegos. La mejora e implementación de la tecnología 3D y los efectos visuales en el audiovisual permite nuevas posibilidades narrativas y artísticas para los creadores, donde los espectadores tienen experiencias de visualización e inmersión cada vez más realistas. Las aplicaciones del 3D en la industria audiovisual permiten la creación de entornos, personajes y efectos que no se podrían reproducir de forma física, hecho que ha aumentado la relevancia tanto del 3D como de los efectos visuales en la producción audiovisual.

En los últimos años, los resultados que se obtienen con los motores en tiempo real se han visto mejorados considerablemente. Su optimización y mejora ha provocado que se empiece a implementar en ámbitos diferentes a los videojuegos como el cine o la televisión.

En el ámbito de los efectos visuales, el renderizado en tiempo real aporta ventajas significativas, por un lado, permite al artista crear y ver en tiempo real el efecto visual que está realizando, facilitando y mejorando la eficiencia en su producción, y por otro lado optimiza los tiempos de renderizado considerablemente, acercando la producción de efectos visuales a usuarios con equipos domésticos.

En este trabajo se propone el estudio y la creación de diferentes efectos visuales, a través de la utilización del motor en tiempo real Unreal Engine 5 con la finalidad de estudiar la viabilidad real de la creación de efectos renderizados en tiempo real.

1.2.- Objetivos

El objetivo principal de este trabajo es estudiar y elaborar efectos visuales en 3D mediante sistemas partículas y fluidos en tiempo real, utilizando *software* 3D y herramientas específicas para la creación de estos, obteniendo unos resultados finales realistas. Para el desarrollo de los diferentes efectos visuales también se plantean unos objetivos secundarios:

El primero de los objetivos secundarios que se plantea es aplicar de forma práctica nuevas tecnologías 3D. El mundo del 3D y los efectos visuales es un mundo en constante desarrollo, y cada año surgen nuevos avances o tecnologías aplicables a la generación de gráficos 3D y efectos visuales. En este TFM se plantea la utilización de alguna de estas tecnologías. Otro objetivo secundario que se propone es desarrollar habilidades en el uso de motores en tiempo real, para así poder abordar los diferentes efectos visuales. Por otro lado, también se plantea el objetivo de analizar la evolución histórica de los efectos visuales, con el fin de elaborar una perspectiva histórica general, la cual haga de base para poder desarrollar los efectos visuales planteados. En último lugar, se propone la creación de diferentes escenarios en ·3D para crear un contexto donde integrar los diferentes efectos visuales, y así poder generar también animaciones y cinemáticas de mayor calidad.

1.3.- Metodología

Para la elaboración de este trabajo primero se han recopilado datos sobre la creación de los efectos visuales y su evolución histórica, apoyándose tanto en documentación oficial de los diferentes programas que se estudian, como en documentación académica de efectos visuales como artículos académicos, libros o páginas webs relacionadas con el estudio de los VFX[1]. También se han identificado los nuevos avances tecnológicos y tendencias que surgen en el sector de los efectos visuales, y, por otro lado, se ha estudiado el funcionamiento de los motores en tiempo real, los sistemas de partículas y sus diferentes utilidades, con el fin de realizar una posterior aplicación práctica de los mismos en un entorno 3D.

Partiendo del desarrollo de un marco teórico como base, se ha diseñado un flujo de trabajo a seguir, para después generar los diferentes efectos visuales en el *software* Unreal Engine 5. Esto se ha realizado con el fin de experimentar de una forma práctica el proceso de producción de algunos efectos visuales en los cuales se utilizan sistemas de partículas y fluidos, así como su integración en un espacio 3D. Para realizar los diferentes efectos visuales, se ha recurrido a documentación oficial y a diferentes foros y tutoriales relacionados con el *software* Unreal Engine.

[1] VFX. Abreviación común de Visual Effects

En este trabajo final de Máster, se ha elegido utilizar un motor en tiempo real para la generación de efectos visuales en ved de otros programas como Houdini o Maya, los cuales son más habituales en la producción profesional, principalmente por motivos de rendimiento, velocidad de procesamiento y el ordenador que se ha utilizado, el cual se detalla en los anexos (ver Anexo 1). También se ha terminado utilizando Unreal Engine 5 debido a que tiene herramientas específicas e intuitivas para generar efectos visuales. Por otro lado, encontramos que este *software* tiene integradas nuevas tecnologías que resultan muy interesantes de aplicar, las cuales se exponen también en este trabajo.

Para explicar el proceso de producción de los efectos visuales propuestos se han creado cuatro fases en las cuales se explica cómo se ha construido el set de la escena, como se han generado los efectos visuales y aspectos del renderizado. Posteriormente se ha hecho una fase de postproducción con los renderizados obtenidos, donde se han editado los diferentes planos, y se ha corregido el color aplicando también diferentes conocimientos aprendidos durante el máster. En último lugar se han elaborado unas conclusiones, donde se comentan los diferentes problemas que han surgido y los resultados obtenidos.

1.4.- Estructura de la memoria

La estructura de este trabajo final de Máster consta de cinco apartados y los anexos. En primer lugar, se ha redactado un apartado introductorio donde se explica la motivación, los objetivos, la metodología y la estructura del trabajo. Después se ha continuado con un segundo apartado, donde se ha elaborado un marco teórico en el cual se estudia la evolución de los efectos visuales, se hace un acercamiento teórico a los sistemas de partículas y fluidos y se tratan conceptos básicos del renderizado en tiempo real. A continuación, se ha elaborado un tercer apartado donde se explican los efectos visuales que componen el TFM, en este apartado también se ha recopilado información sobre motores en tiempo real y se ha diseñado un flujo de trabajo para la realización de los diferentes efectos. Después se ha elaborado un apartado de conclusiones donde se hacen las valoraciones finales y se exponen las dificultades que se han tenido. En último lugar, se ha elaborado un apartado con la bibliografía utilizada y se han añadido los diferentes anexos.

2.- Marco Teórico / Contexto

En primer lugar, se ha elaborado un apartado de contextualización para sentar ciertas bases antes de generar los diferentes VFX. En este apartado se elabora una definición para los efectos visuales y se explica la evolución histórica de los efectos visuales. También se ha realizado una contextualización de los sistemas de partículas y fluidos, ya que van a ser el punto de partida para realizar la mayoría de los efectos propuestos. En último lugar, se han expuesto ciertos conceptos básicos sobre los motores en tiempo real.

2.1.- Efectos visuales. Definición y conceptos clave

Los efectos visuales han tenido siempre relevancia en la industria del cine, pero durante los últimos años su importancia y presencia en el mundo audiovisual se ha visto multiplicada exponencialmente. El progreso tecnológico y las mejoras en el campo del CGI[2], junto la demanda por parte del público de series y películas de ciencia ficción, fantasía o superhéroes ha resultado en que los estudios de cine y compañías de televisión aumenten su inversión en estudios de VFX, ya que son conscientes de que así pueden atraer a un público más amplio y generar mayores beneficios.

Los efectos visuales son definidos como técnicas y procesos utilizados en la industria audiovisual para generar imágenes o escenas que no pueden ser grabadas en la vida real. Estos efectos son utilizados para mejorar o modificar la apariencia de una escena, para crear ambientes o simular escenas imposibles. Para Jeffrey A. Okun y Susan Zwerman, los VFX son definidos como “el término utilizado para describir cualquier imagen creada, alterada o mejorada para una película u otro medio en movimiento, que no se puede lograr durante el rodaje de acción en vivo” (Okun, Zwerman et al., 2010, p2). Para seguir definiendo los efectos visuales, cabe comentar su diferencia con los efectos especiales ya que a menudo existe una creencia en el público general de unir ambos conceptos. Según Eran Dinur, en la era pre-digital, antes de que los efectos fueran en su mayor medida computados, se usaban principalmente trucos ópticos y maquetas lo cual producía que no hubiera una clara

[2] CGI. Computer Generated Imagery

separación entre efectos especiales y efectos visuales, pero hoy en día los términos significan cosas muy diferentes. Los efectos especiales (SFX) son efectos reales y prácticos capturados en cámara durante el rodaje, mientras que los efectos visuales (VFX) son manipulaciones digitales o creación de gráficos por ordenador que son desarrollados principalmente en la fase de postproducción de un proyecto. (Dinur, 2017).

Otro concepto que ayuda a definir a los efectos visuales es el CGI, que hace referencia a trabajos de arte gráfico que han sido generados por una computadora mediante *software* especializado. Según María Díaz, “Este tipo de gráficos se origina mediante un proceso de cálculos matemáticos sobre entidades geométricas tridimensionales producidas en un ordenador, y cuyo propósito es conseguir una proyección visual en dos dimensiones” (Díaz, 2010, p. 206). Por lo tanto, podemos considerar a los VFX como otra fase en la elaboración del CGI.

La simulación de fluidos ya sean líquidos o gases, es uno de los campos dentro de los efectos visuales el cuál ha experimentado un progreso significativo en los últimos años. Este progreso es responsable de que actualmente se puedan realizar simulaciones en ordenadores domésticos, que aunque teniendo tiempos de renderizado bastante elevados, ahora son posibles. Diferentes avances como los renderizados en tiempo real o la aceleración por GPU[3], optimizaciones de los sistemas de partículas y texturizado, ponen al alcance de los artistas *VFX* nuevas posibilidades que antes no podían computar.

La utilización de los motores en tiempo real para la creación de *VFX* aportan diferentes ventajas, aunque también algunas desventajas. En primer lugar, se optimizan los tiempos de renderizado notablemente, provocando que sea posible la computación de estos para usuarios con equipos domésticos u ordenadores con componentes para videojuegos. La utilización de los motores en tiempo real para producir *VFX* también mejoran la eficiencia de la realización de estos permitiendo su visionado en su proceso de creación. Por otro lado, existe una desventaja principal, la cual reside en que todavía no se puede obtener la misma calidad de renderizado que se obtiene usando motores de renderizado tradicionales.

[3] GPU. Unidad de procesamiento gráfico, tarjeta gráfica.

2.2.- Historia y evolución de los efectos visuales

Los efectos visuales han sido usados en el cine desde sus primeras décadas. En los inicios del cine no tardarían en surgir técnicas como el trucaje y la superposición con el objetivo de crear experiencias visuales que impresionaran al espectador. Estas técnicas, con el paso de los años y la llegada de la era digital irían evolucionando hasta hoy en día donde ya prácticamente se pueden generar efectos visuales fotorrealistas.

El primer ejemplo de un efecto visual lo encontramos en un corto llamado “The Execution of Mary, Queen of Scots” (Clark, 1895) donde Alfred Clark, para grabar la escena de una ejecución en los estudios de Thomas Edison, dio con la idea de parar la cámara, e intercambian al actor por un maniquí, y de esta forma continuar con el metraje (Okun, Zwerman et al., 2010). A este efecto conocido como el trucaje por sustitución también llegarían cineastas como George Méliès el cuál descubrió esta técnica cuando su cámara se atascó y la volvió a poner en funcionamiento segundos después comprobando que había diferentes elementos habían desaparecido del plano, siendo substituidos por otros que ahora ocupaban la escena. Méliès en su película “Le voyage dans la lune” (Mèliès,1902), también aplicaría diferentes técnicas predecesoras de lo que hoy conocemos como efectos visuales, como pueden ser la utilización de maquetas, la superposición o las múltiples exposiciones (Mártinez, 2018).

Muchos otros cineastas continuarían desarrollando distintos tipos de efectos, ejemplo de ello es Edwin S. Porter que para su película “The Great Train Robbery” (Porter,1903), dejó las ventanas de la oficina sin exponer utilizando el negro mate, para posteriormente exponerlas con una imagen diferente y así simular un exterior en ellas (Okun, Zwerman et al., 2010). También Segundo de Chomón desarrolló técnicas precursoras como la introducción de maquetas en sus películas. En “El hotel eléctrico” (Segundo de Chomón, 1905) mediante la técnica del paso de manivela consiguió filmar fotograma a fotograma, de forma que podía ir moviendo los elementos creando una ilusión de movimiento. Durante esta película Segundo de Chomón crearía la técnica de la *pixiliación*[4] y sentaría las bases de lo que sería el posterior *stop-motion*[5].

[4] Pixilación. Técnica de animación en la que se usan fotografías de actores y se animan como marionetas.
[5] Stop-Motion. Técnica de animación que consiste en simular movimiento de objetos estáticos mediante una sucesión de fotografías.

Durante los años siguientes se siguieron perfeccionando técnicas como la doble exposición o la realización de maquetas cada vez más realistas. Un ejemplo de este progreso pueden ser películas como “King Kong” (O’Brien, 1933) donde se usó una miniatura de un King Kong para realizar las animaciones y también se siguieron mejorando los *matte paintings* utilizados como fondos que creaban profundidad (Martínez, 2018).

No sería hasta la década de los 50’s cuando John Whitney creó su empresa Motion Graphics Inc. y empezó a generar los primeros efectos de luz analógicos por ordenador (Wells, 2009). Alfred Hitchcock para su película “Vertigo” (Hitchcock, 1958) se fijaría en la empresa de Whitney, y junto con el diseñador gráfico Saul Bass, generarían la primera animación por computadora. Esta primera animación CGI fue la secuencia de créditos de la película (Figura 1), para la cual generaron espirales que representan ecuaciones paramétricas, con una computadora mecánica que había servido en la segunda guerra mundial para la orientación de sistemas antiaéreos.

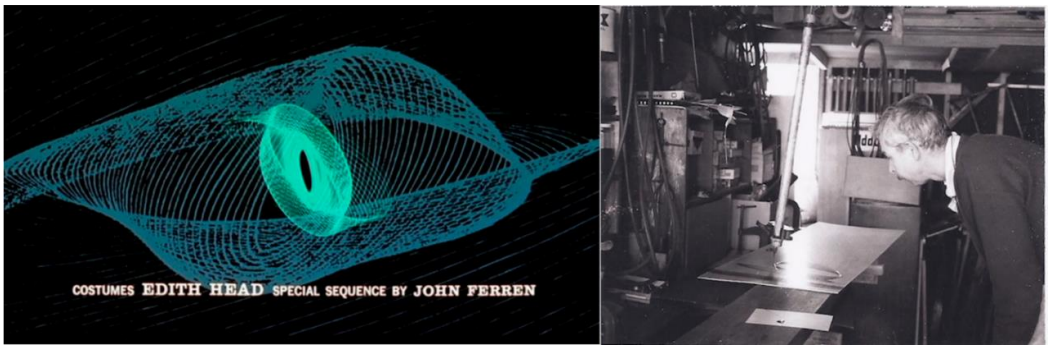

Figura 1. Secuencia de créditos de Vertigo y computadora mecánica.

Pocos años después, en 1961 Ivan Sutherland inventaría el *Sketchpad*, la primera interfaz gráfica con la que se podía dibujar en un ordenador. Esta interfaz abriría nuevas puertas para el diseño en computadora e incluso se implementaron funcionalidades como duplicar un objeto, hacer zoom o borrar líneas. Otra mejora significativa llegaría en el año 1964 con la creación de la primera grabadora de video digital, con la que John Stehura creo su cortometraje “Cibernetik 5.3” (Stehura, 1965), este cortometraje se realizó sin que su creador pudiera ver el resultado, hasta que

utilizó la grabadora de General Dynamics en San Diego, Estados Unidos (Wells, 2009). Durante esta primera parte los años 60 también surgieron las primeras simulaciones, William Fetter para la compañía Boeing crearía el primer simulador de vuelo.

Los efectos visuales volverían a adquirir popularidad con la creación de "2001: A Space Odyssey" (Kubrick,1968). Para esta película se recreó el espacio de la forma más realista que se había visto hasta la fecha, gracias a modelos en miniatura de naves espaciales muy detallados, proyección trasera, modelos de tamaño real y diferentes técnicas como sets rotativos o cámaras inventadas para la película. No obstante, no sería hasta el año 1973, cuando se integró por primera vez una escena CGI en una película. En la película "Westworld" (Crichton,1973) se realizó un efecto que pixelaba la imagen para simular la vista de un androide.

Nuevos avances llegarían de la mano de George Lucas, el cual coordinó los primeros equipos de artistas de efectos visuales para la película "Stars Wars. Episode IV: A New Hope" (Lucas, 1977) los cuales más tarde formarían ILM (Industrial Light & Magic). Fue la película con más efectos visuales hasta la fecha, incluyeron multitud de maquetas a escala real como en miniatura, utilizaron el *chroma key*[6] *y* la composición óptica, e inventaron un sistema de *motion control*[7] llamado *Dykstraflex*, el cual seguía la dirección de un objeto automáticamente (Martínez, 2018).

En los años siguientes seguiríamos viendo un uso extendido de los efectos visuales, películas como "Superman" (Donner,1978), "Alien" (Scott,1979) o "Blade-Runner" (Scott,1982) son ejemplos de ello. Sin embargo, no veríamos la primera aplicación del CGI como lo conocemos hoy en día, hasta la creación de la película "Tron" (Lisberger,1982) por parte de Disney. Para esta película se generó por ordenador un total de 15 minutos de metraje, además de múltiples decorados. Por otro lado, para la película "Star Trek II: The Wrath of Khan" (Meyer, 1982) se creó en Pixar (el cual todavía era una división dentro de ILM) el "Genesis effect", siendo este el primer efecto visual que utilizaba partículas (Figura 2).

También por parte de la compañía Digital Production se produjeron avances en 1984 reflejados en la película "The Last Starfighter" (Castle, 1984). En esta película se

[6] Chroma key. Técnica de postproducción que consiste en la extracción de un color (verde o azul) para remplazarlo por una imagen o video.
[7] Motion Control. Sistema digital de control de movimiento para cámaras.

crearon imágenes generadas por computadora con una complejidad notablemente superior, hicieron la primera simulación de fluidos y el primer animal en CGI fotorrealista (Okun, Zwerman et al., 2010).

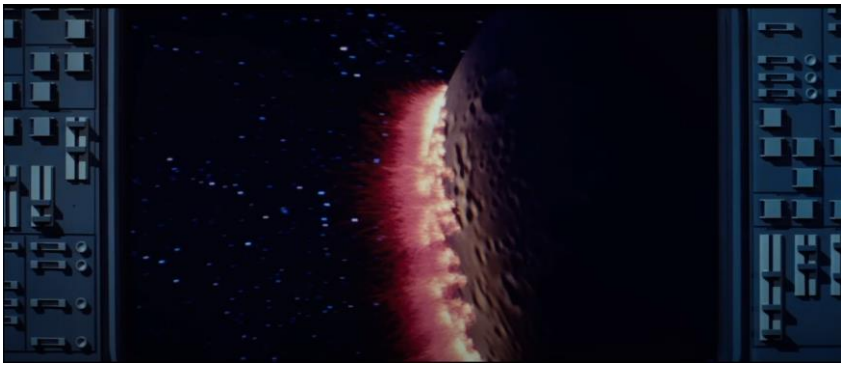

Figura 2. Primer uso de un sistema de partículas en Star Trek II

Durante la segunda parte de la década de los 80 se siguieron produciendo avances como el primer humano digitalmente animado en la película “The Young Sherlock Holmes” (Levinson, 1985), pero estos efectos seguían siendo muy costos de producir y empezarían a cuestionarse. Sin embargo, con la llegada de los 90 y los grandes avances en el campo de la informática, el CGI volvió a cobrar sentido en la industria cinematográfica. Algunas películas de la época como “Terminator II: The Judgment Day” (Cameron, 1991), “Jurassic Park” (Spielberg, 1993) o “Toy Story” (Lasseter,1995) fueron todo un éxito en taquilla consiguiendo que el uso del CGI en el cine se terminara de consolidar en la industria. En “Terminator II” se perfecciono el efecto del *morphing* [8] creando transformaciones realistas de los personajes en robots*.* Según Paul Wells:

> Terminator II: El juicio final demostró que podía utilizarse la tecnología CGI para conseguir buenos resultados estéticos y narrativos, lo que suponía un avance innovador respecto a cualquier obra anterior. Con la estandarización creciente del software requerido, proliferaron los estudios de producción y el sistema CGI se convirtió en una herramienta intrínseca de expresión dentro del sector comercial y del ocio (Wells, 2009, p. 123).

[8] Morphing. Efecto que consiste en la transformación de un objeto en otro por medio de distorsiones y transiciones en base a puntos de referencia.

Figura 3. Efecto morphing en Terminator II

Para la película “Jurassic Park” (Spielberg, 1993) se realizarían las primeras criaturas orgánicas con texturizados notablemente más realistas, aunque el uso del CGI en esta película fue revolucionario, encontramos que para la mayoría de las escenas de dinosaurios utilizaron otras técnicas ya muy perfeccionadas como el *stop-motion* y los *animatronics* [9], el uso del CGI fue relativamente limitado en esta película y utilizado en lo estrictamente necesario.

Pocos años después, Pixar produciría “Toy Story” (Lasseter,1995) que fue la primera película animada completamente en CGI, para poder realizar la película usarían el motor de render “Renderman” el cual llevaban desarrollando desde principios de los 90. Para poder renderizar esta película en los años 90 se tuvieron que utilizar alrededor de 300 ordenadores que renderizaban un *frame* cada 7 horas.

Los años 90 para el CGI concluirían con dos películas que incorporarían nuevos avances como “Titanic” (Cameron, 1997) donde mejoraron los sistemas de fluidos con el fin de incorporar comportamientos más realistas en los océanos, también en película “Matrix” (Wachowski,1999), encontramos avances significativos en efectos visuales como el *bullet-time,* el cual consiste la creación secuencias de acción rotativas ralentizadas.

Con la llegada del nuevo siglo y la tecnología creciendo de forma exponencial, en los 2000 se perfeccionarían técnicas como el *motion capture*[10], donde ya se podían

[9] Animatronics. Animación por medio de maquetas que utilizan mecanismos electrónicos o robóticos.
[10] Motion Capture. Técnica de grabación de movimiento que se realiza para posteriormente reproducirla en un personaje 3D.

alcanzar resultados fotorrealistas. El primer ejemplo lo encontramos en Final Fantasy:The Spirits Within (Sakaguchi, 2001) y posteriormente en el “El señor de los anillos: La comunidad del anillo” (Jackson, 2002) donde crearon las animaciones de Gollum con esta técnica. Junto con los avances que se consiguieron en los *crowds*[11] para simular batallas a gran escala, la compañía Weta en esta película llevaría a un nuevo nivel la producción de CGI y los efectos visuales.

De aquí en adelante, la tecnología y la capacidad de procesamiento seguirían evolucionando permitiendo alcanzar mediante CGI resultados fotorrealistas. También empezarían a surgir multitud de nuevas herramientas y software desarrollado específicamente para realizar efectos visuales. Películas como Avatar o Interstellar continuarían creando efectos visuales cada vez más espectaculares e inmersivos. En definitiva, la evolución y mejora de los efectos visuales los terminaría posicionado como un elemento clave en la producción cinematográfica y televisiva en constante desarrollo y mejora.

2.3.- Contextualización de los sistemas de partículas. Simulaciones de fluidos

Para realizar multitud de efectos visuales como humos, líquidos o explosiones habitualmente se utilizan sistemas de partículas. Un sistema de partículas está conformado por puntos (partículas) a los cuales se asignan diferentes propiedades como una masa, velocidad y una posición, para después hacerlas reaccionar a unas fuerzas que modifican su comportamiento. En otras palabras, un sistema de partículas es un objeto donde existen subobjetos que se comportan en base a unas reglas comunes.

En la mayoría de software 3d, los sistemas de partículas comparten algunas variables o procedimientos básicos los cuales se detallan a continuación:

- Emisor: Las partículas nacen por medio de un emisor o *spawn*, y posteriormente se generan en el espacio de acuerdo a las físicas que se establecen en la simulación. Normalmente se define un *spawn rate,* el cual

[11] Crowds. Simulaciones de multitudes de agentes o personajes en 3D.

indica el número de partículas que se emiten.

- *Lifetime*: Hace referencia al ciclo de vida de las partículas. El tiempo de vida que tarda en morir una partícula.
- Velocidad: Variable que hace referencia a la dirección de las partículas y al tiempo que tardan en hacer su recorrido (velocidad). Es usual direccionar las partículas dando valores de velocidad en los diferentes ejes x,y,z.
- Posición: Variable referente a la posición en el espacio tridimensional de las partículas emitidas. Cada punto tienen su valor en los ejes x,y,z.
- Tamaño inicial: Esta variable define la escala inicial de las partículas, suele ser habitual definir un valor máximo y uno mínimo, para que las partículas se generen en diferentes tamaños de acorde a estos límites.
- Forma: Esta variable define los límites de la región donde nacen las partículas, pueden ser formas geométricas como una esfera o un cubo, dentro de las cuales nacen las partículas.

Los sistemas de partículas son una solución muy efectiva cuando se busca reproducir fenómenos naturales como el humo o el fuego, los cuales se comportan de forma irregular. William T. Reeves en su artículo “A Technique for Modeling a Class of Fuzzy Objects” describe como los sistemas de partículas pueden representar un objeto difuso, formado por un conjunto de partículas con sus propias características. También explica cómo estos sistemas pueden responder a comportamientos físicos, como a diferentes fuerzas que influyan en su comportamiento, además teoriza varios algoritmos para controlar el aspecto y comportamiento de los sistemas. (Reeves, 1983)

A día de hoy, encontramos que los sistemas de partículas son una herramienta básica en cualquier *software* ·3D, aparte de para reproducir fenómenos naturales, también tienen multitud de aplicaciones, como por ejemplo la creación de pelo o tejidos. Para realizar efectos visuales con los sistemas de partículas, se puede utilizar las partículas como tal, o se pueden substituir estas por geometrías más complejas o imágenes 2D. Para crear efectos y simulaciones más realistas, las partículas se hacen reaccionar a diferentes fuerzas como la gravedad, viento o turbulencias. La configuración de

turbulencias es una forma muy habitual de resolver algunos efectos, donde se necesita que las partículas tengan una imprevisibilidad estructurada. (Okun, Zwerman et al., 2008).

A la hora de realizar las simulaciones podemos computarlas por medio de la CPU o la GPU, ambos procedimientos son correctos, pero encontramos algunas diferencias en su resultado que pueden ser utilizadas dependiendo del uso que se le quiera dar al sistema de partículas. La principal diferencia es el número de partículas que podemos emitir, usando la GPU se puede obtener un número mayor de partículas, sin embargo, hay características que se le otorgan a las partículas como materiales emisivos, los cuales no son compatibles y se deben calcular mediante CPU.

La recreación de fenómenos naturales siempre ha sido un área de gran importancia dentro de los efectos visuales, y los fluidos siempre han resultado un desafío por la cantidad de elementos o partículas que interactúan entre sí, los cuales la máquina necesita calcular para poder reproducir (Figura 5).

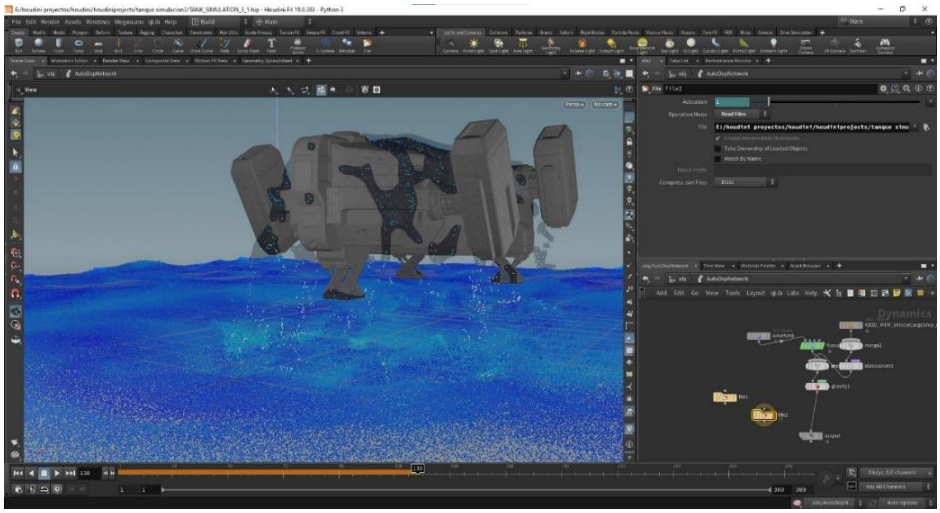

Figura 5. Sistema de fluidos generado con partículas en Houdini

Para la física, un fluido es una substancia que continuamente se deforma y fluye bajo una tensión, por lo tanto, no tienen una forma fija y se adaptaran al recipiente que lo contenga. Para reproducir fenómenos naturales como un océano, las aplicaciones 3D utilizan sistemas de fluidos los cuáles crean el movimiento utilizando algoritmos que usan como base las ecuaciones Navier-Stokes. Estas ecuaciones, describen el

comportamiento de varios fluidos (incluidos los gases) y no dan como resultado un número concreto, sino la velocidad en la cual se mueven las partículas (Seymour, 2016).

Para poder crear movimiento en una simulación de fluidos hay dos posibilidades, por un lado, tenemos el método lagrangiano (Lagrangian viewpoint) y el euleriano (Eulerian viewpoint). El enfoque lagrangiano representa el movimiento como un sistema interpolado con un determinado número de partículas. Por otro lado, el enfoque euleriano que utiliza una cuadrícula fija donde observa puntos fijos en el volumen y mide como las cantidades de fluido cambian en el tiempo. (Bridson, 2016). Actualmente, las empresas desarrolladoras de software 3d implementan métodos mixtos. En aplicaciones especializadas 3D como por ejemplo Houdini o Unreal Engine utilizan el método FLIP (Fluid-Implicit-Particle) para las simulaciones de líquidos. El método FLIP es un sistema hibrido entre los sistemas de partículas y los volúmenes. Según Bridson, cuando los fluidos FLIP son resueltos, se crea un campo de velocidad temporal. Las velocidades de las partículas se transfieren a la rejilla y esta se utiliza para realizar la proyección del fluido. Debido a esto, las partículas no se amontonan unas encima de otras ni se mueven todas en la misma dirección (Seymour, 2016). En la documentación oficial de Unreal Engine, se nos remite a que en *Niagara Fluids*[12] implementan este algoritmo para simular líquidos (Figura 6).

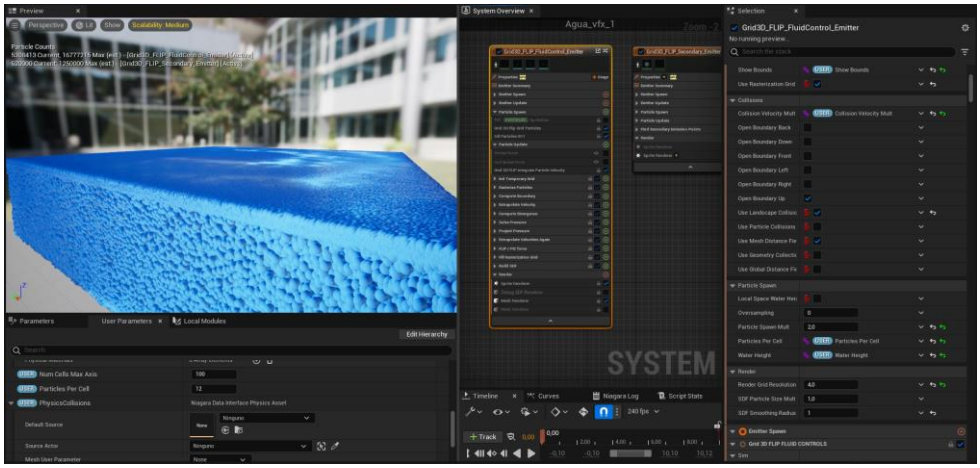

Figura 6. Sistema de fluidos con partículas en Unreal Engine

[12] Niagara Fluids, es un plugin nativo de Unreal Engine el cual se utiliza para la creación de VFX.

2.4.- Renderizado en tiempo real

El renderizado es un proceso en el cual se genera una imagen o video bidimensional a partir de un modelo 2D o 3D. Durante este proceso, el ordenador principalmente calcula la geometría del modelo, el texturizado y el comportamiento de la luz. Este proceso se lleva a cabo por medio de motores de renderizado, los cuales procesan todos los datos dando como resultado una imagen o video.

Muchos motores de render vienen incluidos en el *software* 3D como por ejemplo Cycles en Blender[13] o Arnold en Maya[14], pero también existen multitud de motores externos compatibles con diferentes programas como Redshift o RenderMan. Todos ellos, generan simulaciones con comportamientos realistas y cuentan con algoritmos propios que mejoran continuamente para poder diferenciarse, sobre todo en cuestiones de rendimiento.

Actualmente encontramos dos tendencias en cuanto a renderizado ya establecidas:

- Renderizado offline: Es el renderizado tradicional el cuál se ha estado utilizando para producir de manera estandarizada *VFX* y 3D en cine. Su principal ventaja es el alto grado de fotorrealismo que se puede alcanzar, para ello se utilizan técnicas como la utilización de *raytracing* [15] para calcular con más fidelidad el comportamiento de la luz y una mayor cantidad de reflexiones y rebotes de esta. Por otro lado, este renderizado requiere de tiempos mucho más largos para su procesado, y su uso está restringido a contenido no interactivo por lo que no se puede utilizar en videojuegos. Durante los últimos años, se ha estado implementando el uso de la aceleración por GPU con el fin de acortar los tiempos de *render*, ejemplos de estas implementaciones las encontramos en motores como Cycles (Blender) o Karma (Houdini[16]) los cuales ya procesan imágenes utilizando el CPU y el GPU.
- Renderizado en tiempo real: En contraposición a la forma de renderizado tradicional, los motores de renderizado en *real-time* calculan las imágenes con mucha más velocidad, de forma que las imágenes se perciben de forma

[13] Blender. Programa informático para generar gráficos 3D que abarca diferentes áreas como el modelado, texturizado o iluminación entre otros.
[14] Maya. Programa informático para modelado y animación 3D, desarrollado por Autodesk
[15] Raytracing. Trazado de rayos, algoritmo para renderizar de forma realista una iluminación en una escena 3D.
[16] Houdini. Programa informático 3D especializado en la generación de efectos visuales

instantánea. Como es lógico los cálculos de los motores en tiempo real no tienen la calidad que se puede alcanzar con un renderizado tradicional, pero el constante desarrollo ha permitido que los resultados de estos motores se acerquen significativamente a los obtenidos con métodos de renderizado tradicionales. Ejemplos de motores en tiempo real los podemos encontrar en Unity[17] o Unreal Engine[18]. Este método de renderizado es habitualmente utilizado para videojuegos o realidad virtual, pero debido a su desarrollo en los últimos años ya se están incorporando en producciones cinematográficas o televisivas.

En el cine, los renderizados en tiempo real también se han implementado durante los últimos años. Esta tecnología se utiliza para poder crear escenarios donde se reproducen efectos visuales, físicas o diferentes animaciones en tiempo real. Para poder reproducir estas animaciones se utilizan pantallas LEDs[19] de alta resolución, las cuales interactúan con las cámaras del *set* y el motor de *real-time* que se esté utilizando. Para grabar "The Mandalorian" (Favreau, 2019), ILM utilizó la tecnología *Stagecraft* (Figura 7) la cual usa pantallas compuestas por diferentes nodos que reproducen el escenario·3D y también iluminan a los diferentes personajes.

Figura 7. Set de produción virtual para The Mandalorian

[17] Unity. Motor de videojuegos desarrollado por Unity Technologies.
[18] Unreal Engine. Motor de videojuegos desarrollado por Epic Games.
[19] Pantallas LED. Pantallas que funcionan con diodos emisores de luz.

En televisión también se utiliza este tipo de motores para la creación de platós virtuales o para realizar gráficos o visualizaciones 3d, mejorando la calidad y optimizando los tiempos de producción de cualquier elemento en 3d que se necesite realizar.

Un motor de renderizado, lo podemos entender también como un conjunto de algoritmos que permiten la reproducción y exportación de diferentes imágenes. En un proceso de renderizado, podemos distinguir diferentes fases las cuales están ordenadas y organizadas en un *pipeline*:

La primera fase hace referencia al procesamiento de datos, para esto el motor recopila y procesa toda la información de la escena. Entre esta información encontramos geometría, iluminación, texturas, y cámaras principalmente todo aquello que tenemos en escena. El motor almacena toda esta información y la divide en estructuras de datos para pasarla a la siguiente fase.

Después hay una segunda fase de geometría, en ella se calcula la información referente a los polígonos, su comportamiento y ubicación en la escena. Entre algunos procesos que ocurren en esta fase encontramos que se calculan las transformaciones geométricas en el espacio, se aplica la iluminación y se calculan las sombras, también se realiza la proyección, que consiste en la conversión de la geometría en un espacio 3D a una imagen 2D por medio de sistemas de mapeo. Por último, también ocurren procesos como el *clipping*, que consiste en encontrar aquellas partes de la geometría que no son visibles en la escena y desecharlas con el fin de optimizar el procesado de las siguientes fases.

En la tercera fase, se produce la rasterización, en esta fase se convierte toda la información que sale en escena en píxeles a los cuales se les otorgan sus valores correspondientes.

Por último, en la cuarta fase se aplican procesos de post-procesado. Entre ellos encontramos diferentes procesos como el *anti-aliasing*[20], *motion-blur*[21] o el *bloom*[22], que ayudan a mejorar la imagen previamente calculada y darle un acabado realista.

Las diferentes fases que constituyen un *pipeline* de renderizado en tiempo real

[20] Anti-aliasing. Efecto de postprocesado de imagen que reduce los dientes de sierra o pequeñas distorsiones que aparecen en los bordes de las geometrías al renderizarse.
[21] Motion-blur. Efecto de postprocesado que recrea el desenfoque de movimiento que percibimos los humanos.
[22] Bloom. Efecto de postprocesado que aumenta la luminosidad de las fuentes de luz.

contribuyen en su totalidad a los resultados finales que el artista obtiene, y en ellas las empresas especializadas en la industria del 3d implementan nuevas funcionalidades y algoritmos que mejoran y optimizan su funcionamiento. Sin embargo, cada pipeline suele variar entre los diferentes motores o sistemas de renderizado.

Otros aspectos esenciales que ayudan significativamente mejorar la calidad de los renderizados, ya sean en tiempo real o no, son la iluminación y el texturizado.

La iluminación en un renderizado 3d juega un papel muy importante ya que es uno de los factores que permite crear una sensación de realismo en las escenas generadas. En los motores *real-time* se incluyen simulaciones de efectos de luz y sombras dinámicas, las cuales interactúan con los demás objetos al instante. La iluminación en los motores de tiempo real se divide en dos categorías.

Por un lado, tenemos la iluminación global, la cual es referente a la iluminación ambiental que se aplica a toda la escena, generando un ambiente y unas sombras generales. Gracias a los sistemas de iluminación global se puede generar iluminación indirecta, que es la que se produce al rebotar la luz en las diferentes superficies, que a su vez pueden reflejar otras superficies (Shirley, 2009). Estos sistemas también permiten la iluminación directa, provocados por fuentes de luz que directamente enfocan a los objetos.

Para poder calcular la luz, los motores de renderizado utilizan algoritmos que resuelven diferentes procesos como la radiosidad, el *Ray Tracing* o el *Ambient Occlusion* [23] entre otros, los cuales se detallan a continuación. La radiosidad es una técnica desarrollada para simular los rebotes de la luz en superficies difusas. En este algoritmo se utilizan ecuaciones para calcular todas las superficies del entorno y como se propaga la luz a través de esta teniendo en cuenta sus interreflexiones (Haines, 2018). Por otro lado, la técnica del *Ray Tracing* rastrea los rayos de luz desde el punto de vista de la cámara a la fuente de luz. En esta técnica se calcula las reflexiones de luz que percibimos los humanos, lo cual es un elemento diferenciador de los algoritmos de radiosidad que calculan todo el entorno ofreciendo una iluminación global. Los motores de renderizado actuales disponen de algoritmos para llevar a

[23] Ambient Occlusion. Oclusión ambiental. Método de sombreado que añade realismo a las reflexiones producidas por una fuente de luz.

cabo ambos procesos, siendo las técnicas de radiosidad más óptimas para calcular reflexiones en las superficies difusas, mientras que, por otro lado, con las técnicas de Ray Tracing se obtienen mejores resultados en las reflexiones especulares.

Otra técnica utilizada en la iluminación 3D es el Ambient Occlusion, los algoritmos que son utilizados en esta técnica generan un mapa de oclusión los cuales determinan que zonas de la escena deben ser más oscuras. Habitualmente se utiliza con mayor frecuencia para interiores donde hay menos luz y hay más objetos angulosos (como esquinas), los cuales provocan que la luz se atenúe y forme sombras.

La segunda categoría en los sistemas de iluminación, es la iluminación local la cual hace referencia a fuentes de luz concretas como pueden ser las luces de una habitación, el sol o un material emisivo. Mediante estos sistemas se crean luces con sobras y detalles precisos. En la iluminación local, la luz índice directamente sobre la superficie y esto provoca que se acentúen las diferencias entre luces y sombras.

Para determinar la apariencia de un objeto renderizado, se utilizan los *shaders* (sombreadores) los cuales describen las variaciones de los colores en función de factores como la superficie, la orientación o la iluminación (Haines, 2018). El *shading* también determina cómo interactúan las diferentes luces y sus rebotes, según el *software* 3D que se utilice el término *shading* también puede introducirse como material. Para crear los diferentes materiales de nuestro modelo se suele recurrir al uso de texturas. El texturizado hace referencia al proceso en el que se aplican texturas a un modelo 3D. Estas texturas pueden ser imágenes como fotos o pueden generarse proceduralmente[24]. Para la realización de texturas cabe comentar que existe software especializado para ello como Substance Painter[25], aunque la mayoría de *software* para modelado 3D como puede ser el caso de Blender o Zbrush[26] llevan incorporados sistemas para poder pintar texturas o realizarlas proceduralmente.

Habitualmente se utiliza el mapeo de UVs, técnica la cual consiste en asignar coordenadas de una textura a los polígonos de nuestro modelo. Con este método podemos proyectar diferentes mapas de texturas en la parte de la geometría que nos interesa. Para dotar a los materiales de diferentes propiedades es usual utilizar

[24] Procedural. Generación por procedimientos habitualmente de forma nodal o mediante código.
[25] Adobe Substance Painter. Software para realizar texturas y materiales 3D.
[26] Zbrush. Software 3D especializado para modelar, esculpir o texturizar.

diferentes mapas de texturas. Entre los diferentes tipos de mapas encontramos:

- *Diffuse map* (Albedo): Mapa el cual contiene la información del RGB, define el color base.
- *Metallic*: Mapas que otorga propiedades metálicas a los materiales, define las reflexiones de la luz en material.
- *Roughness*: Mapas que definen la rugosidad del material afectando a los rebotes de la luz.
- *Specular*: Mapas que definen la suavidad del material, afectando al comportamiento de la luz.
- *Normal map*: Mapas de normales los cuales simulan la profundidad del material.
- *Height map:* Mapa que define el volumen del material.
- *Ambient Occlusion* (AO): Mapa que define el comportamiento de las sombras y el volumen.

Una de las tendencias actuales para el texturizado de modelos 3D, es la utilización de materiales PBR (Physically Based Rendering). Los materiales PBR se emplean para crear representaciones más realistas basadas en aspectos físicos de la luz y los materiales y se utilizan tanto para la creación de texturas fotorrealistas como para la creación de materiales estilizados. Estos materiales se crean mediante la utilización de diferentes mapas de texturas, como los que se pueden observar en la figura 8.

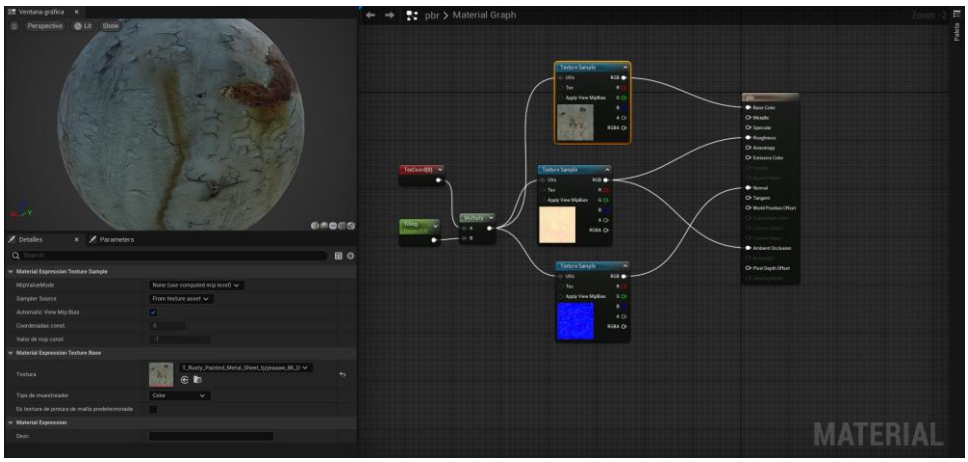

Figura 8. Material PBR en Unreal Engine

3.- Realización de casos prácticos

En este apartado, en primer lugar, se ha recopilado información sobre motores de renderizado en tiempo real que contienen herramientas para la producción de efectos visuales, con el objetivo de escoger una herramienta para generar los diferentes efectos visuales propuestos. Después, se ha diseñado un flujo de trabajo para poder aplicarlo a los diferentes proyectos, los cuales se detallan en este apartado. Para explicar cómo se han realizado los proyectos se han expuesto cuatro fases para cada uno de ellos: En la primera de ellas, se explica cómo se ha configurado los escenarios, después, en una segunda fase, se explican los diferentes efectos visuales que contienen las escenas. A continuación, se expone una tercera fase donde se tratan cuestiones del renderizado, y por último se ha elaborado una cuarta fase donde se realiza la composición y corrección de color de la escena.

3.1.- Motores en tiempo real y aplicaciones 3D

Hasta hace relativamente pocos años, la industria cinematográfica había confiado únicamente en los motores tradicionales para la generación de su *CGI*. Debido a las mejoras de rendimiento y calidad acontecidas en los últimos años, ha empezado a ser viable la producción de *CGI* y efectos visuales con motores de renderizado en tiempo real en cine, algunas aplicaciones exitosas de estos motores las podemos ver en “Westworld” (Nolan, 2016) o “The Mandalorian” (Favreau, 2019). Por otro lado, en el ámbito de los videojuegos los motores en tiempo real son una herramienta necesaria y totalmente estandariza. En este apartado se exponen algunos de ellos.

3.1.1.- Unity

Unity es un motor de videojuegos creado por la compañía Unity Technologies, el cual fue lanzado en la conferencia mundial de desarrolladores de Apple en 2005 obteniendo éxito y fondos para su desarrollo. En sus primeros años, Unity Technologies se enfocó en el desarrollo de juegos para el sistema operativo Mac OS [27], sin embargo, pronto darían soporte a otros sistemas operativos como

[27] Mac OS. Sistema operativo desarrollado por Apple para sus computadoras Macintosh.

Windows[28] y Linux[29].

En el año 2008, con el lanzamiento de Unity 2.0, se introdujeron diferentes mejoras como el soporte 3d para dispositivos móviles, convirtiéndose en una de las primeras plataformas que daban soportes para el desarrollo de juegos en el sistema operativo iOS[30]. Unos años más tarde empezarían a dar soporte también para la creación de juegos en Android[31] y se convertiría en una herramienta muy popular para los estudios pequeños o desarrolladores independientes. En el año 2009, la compañía decidiría convertir su versión del software *indy* en gratuita, consiguiendo que su uso prácticamente se estandarizara en estudios independientes de videojuegos. En el año 2012 y con la llegada de Unity 4, el motor empezaría a dar soporte a consolas de Sony, Microsoft y Nintendo (firmaron acuerdos de compatibilidad) y empezaría a utilizarse en estudios profesionales. Una vez completamente integrado en la industria de los videojuegos, Unity ha seguido mejorando hasta convertirse en una herramienta muy versátil, con la que crear videojuegos o renderizados en tiempo real (Figura 9).

Figura 9. Interfaz de Unity.

El motor de Unity utiliza el lenguaje de programación C#[32] para la creación de sus *scripts*[33], y cuenta con una interfaz más simple que otras aplicaciones como Unreal Engine. También tiene una curva de aprendizaje bastante asequible, ya que existe

[28] Windows. Sistema operativo desarrollado por Microsoft.
[29] Linux. Sistema operativo de código abierto desarrollado por su propia comunidad de usuarios.
[30] iOS. Sistema operativo móvil desarrollado por Apple para el iPhone.
[31] Android. Sistema opertivo móvil basado en Linux.
[32] C# (C Sharp). Lenguaje de programación desarrollado por Microsoft diseñado a partir de los lenguajes C y C++.
[33] Script. Término utilizado para designar una secuencia de comandos.

una gran cantidad de documentación online, una amplia comunidad muy activa en foros.

Para la creación de efectos visuales, Unity cuenta con la herramienta “Visual Effect Graph”, la cual permite la creación de todo tipo de efectos visuales de forma nodal. Con esta herramienta podemos simular en tiempo real efectos visuales como partículas, explosiones o humos. Otras funcionalidades que incluye esta herramienta son las compatibilidades con las herramientas de *shading* de motor y la introducción de *scripts* para generar interacciones o dinámicas complejas.

3.1.2.- Unreal Engine

Unreal Engine es un motor de videojuegos desarrollado por la compañía Epic Games. Su origen lo encontramos en 1998, año en el cual desarrollaron la primera versión de este *software*. En esta primera versión el *software* ya se podía renderizar, detectar colisiones o gestionar archivos de forma eficiente. En el año 2002, Epic Games volvió a escribir el código del motor, creando la versión Unreal Engine 2, el cual daba soporte a las consolas del momento como PlayStation 2[34] o Xbox[35]. Años después en 2006, salió la versión 3 del *software* diseñado para la nueva generación de consolas como PlayStation 3 o Xbox 360. En esta nueva versión incluirían nuevos avances técnicos, sobre todo en el área de la iluminación y el texturizado, aunque también se introducirían avances en las simulaciones de físicas y en las aplicaciones de IA. Algunos videojuegos que popularizaron el motor en esta versión fueron Gears of War (Epic Games, 2006) o BioShock (2K, 2008), donde se alcanzaron niveles de calidad sorprendentes para la época.

En el año 2014 Epic Games desarrollaría Unreal Engine 4, el cual un año más tarde se liberaría para que los usuarios pudieran descargarlo gratis con el 100% de sus herramientas. Esto fue posible gracias a la adquisición de la compañía por el gigante asiático de los videojuegos Tencet, empresa la cual compró la mitad de las acciones de Epic Games. En la versión 4 de este software se implementó el uso del *raytracing* en tiempo real. También mejoraron herramientas para realizar animaciones y cinemáticas e introdujeron un sistema de VR. Unreal Engine 4 ya proporcionaba unos

[34] PlayStation 2 Segunda videoconsola desarrollada por Sony.
[35] Xbox. Videoconsola desarrollada por Microsoft.

resultados fotorrealistas, pudiéndose extender su uso en otros ámbitos como la visualización arquitectónica o el cine.

Para la siguiente generación de consolas, Epic games publicaría en 2021 Unreal Engine 5. Una de las novedades que incluye Unreal Engine 5 es la introducción de la tecnología Nanite, un sistema de virtualización de geometría que permite renderizar en tiempo real millones de triángulos. Este sistema de geometría virtualizada permite renderizar detalles a escala de pixel y también mejora el rendimiento cuando se están procesando muchos objetos. Las mallas Nanite también ocupan menos espacio en disco, su algoritmo de compresión permite que una malla con un millón de triángulos ocupe tan solo 14 MB de memoria. Este sistema funciona de forma inteligente dando prioridad a los detalles que se pueden percibir, manejando el nivel de detalle automáticamente teniendo en cuenta la posición de la cámara. Por otro lado, también permite la gestión y renderizado de geometrías complejas, con las cuales se pueden obtener resultados realistas y con multitud de detalles. Entre otras mejoras encontramos la implantación de nuevas herramientas para mejorar las físicas como Chaos Physics, una herramienta que facilita la destrucción de objetos. También han incorporaron mejoras en la iluminación, siendo el caso de la herramienta Lumen. Para generar iluminaciones realistas Unreal Engine utiliza Lumen, un sistema global de iluminación y reflejos dinámico. En la documentación oficial de Unreal Engine, se nos remite a que Lumen genera una interreflexión difusa con rebotes infinitos y reflexiones especulares indirectas en entornos grandes y detallados a gran escala.

Este motor está escrito en C++[36], lenguaje de programación que también utiliza para que los usuarios creen sus *scripts* dentro del motor. Tiene una interfaz algo compleja, pero en su versión 5 se ha optimizado significativamente. Cabe destacar que Unreal Engine tiene gran cantidad de recursos online, cuenta con su plataforma *"Unreal Engine marketplace"* donde hay multitud de *assets*[37]. También está vinculado con Quixel Megascam, que es una librería que cuenta con multitud de modelos 3D y texturas.

Para la creación de efectos visuales en Unreal Engine, el software cuenta con un *plugin* nativo dedicado a la generación de estos. Este *plugin* se denomina Niagara y con él se puede generar en tiempo real multitud de efectos como sistemas de

[36] C++. Lenguaje de programación compilado desarrollado por Bjarne Stroustroup en los laboratorios At&T
[37] Assets. Diferentes recursos como modelos 3D o escenarios.

partículas, fluidos o explosiones en tiempo real. Este *plugin* trabaja de forma modular y permite una gestión de los diferentes atributos que se estén configurando bastante asequible. Por otro lado, cuenta con diferentes ejemplos a modo de plantilla desde los cuales empezar a realizar efectos, así como diferentes materiales.

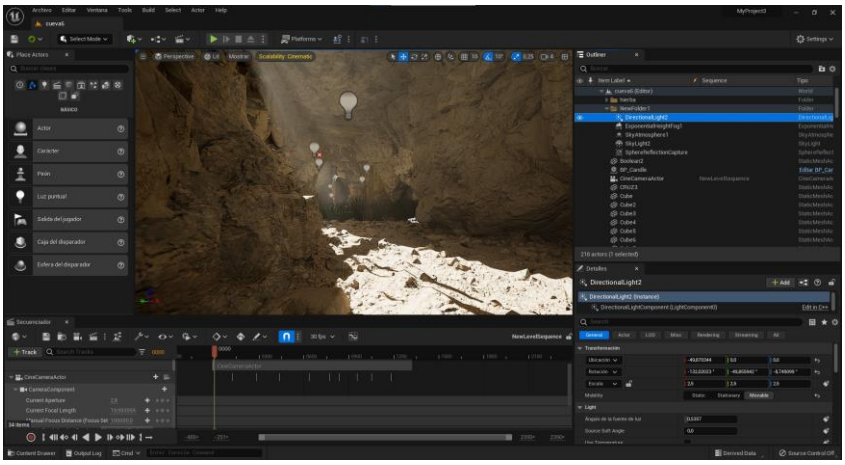

Figura 10. Unreal Engine interface

3.1.3.- Cryengine

Cryengine es un motor de videojuegos desarrollado por Crytek. El motor fue empleado por primera vez en 2002, con el videojuego FarCry (Cyrtek, 2004). Pocos años después fue comprado por Ubisoft, compañía de videojuegos que invirtió en su mejora y desarrollo. Algunos otros juegos desarrollados con este motor son Crysis (Cyrtek & EA, 2007) o Sniper: Ghost Warrior (City Interactive, 2013).

CryEngine, destaca por sus resultados de alta calidad, el motor está centrado en llegar a alcanzar resultados muy realistas. Este *software* contiene herramientas focalizadas en la creación físicas con interacciones muy realistas. También contiene un sistema de iluminación donde la luz puede rebotar tanto en objetos estáticos como dinámicos, además de una oclusión ambiental a gran escala.

Aunque el motor es gratuito desde su versión 5, no existe demasiada información online y su uso no es muy extendido entre desarrolladores independientes. El hecho de que no exista mucha documentación más allá de la oficial puede resultar una barrera de entrada bastante difícil de sortear para usuarios principiantes en motores de tiempo real.

3.1.4.- Justificación del motor utilizado

Para la elaboración de este trabajo se ha optado por utilizar Unreal Engine 5. Este *software* se ha elegido ya que proporciona resultados fotorrealistas y cuenta con herramientas específicas para generación de efectos visuales. Por otro lado, la licencia de este programa es bastante flexible, el motor está disponible de forma gratuita con el 100% de sus funcionalidades. Epic Games también permite la publicación de cualquier contenido generado con su motor, y no pide un porcentaje de las ganancias que se generen con su contenido hasta que estas sean superiores al millón de dólares, caso en el que demandan un 5% de las ganancias.

Entre otros aspectos encontramos que este *software* tiene una interfaz intuitiva y parecida a otras aplicaciones 3D, y además cuenta con herramientas específicas para desarrollar efectos visuales. Por otro lado, existe una gran documentación online sobre el manejo de este programa para la creación de efectos visuales. También encontramos especialmente interesante la herramienta Niagara, por el fácil manejo para añadir atributos a nuestros sistemas y modificarlos.

Este software se ha elegido también debido a que cuenta con nuevas tecnologías que optimizan el rendimiento y mejoran la calidad de los resultados que se pueden obtener, como es el caso de Nanite o Lumen.

Respecto al resto de motores mencionados anteriormente como Unity y Cryengine encontramos, que mediante Unity se pueden obtener resultados similares a Unreal, pero algo menos realistas. También encontramos que la herramienta para realizar efectos visuales de Unity algo menos intuitiva que las herramientas de Unreal. Por otro lado, con el caso Cryengine, encontramos que ofrece resultados al mismo nivel que Unreal Engine, sin embargo, la documentación online y los recursos disponibles para aprender este programa son bastante escasos.

3.1.5.- Otras aplicaciones 3D necesarias

Estos motores gráficos, son dependientes de otras aplicaciones 3D en diferentes procesos como el modelado o el texturizado. Aunque incluyen herramientas para modelar, aun resultan toscas si las comparamos con las herramientas de otros programas específicos para modelar como 3d Max o Blender. Es por eso que es usual utilizar otros programas durante estos procesos. Estos también resultan útiles para

realizar conversiones de formato para los diferentes modelos 3D con los que se trabaja en el motor de tiempo real.

Habitualmente los motores en tiempo real trabajan con formatos como FBX, OBJ o Alembic, formatos que además de la geometría pueden contener otra información como texturas, o datos de animaciones como es el caso del FBX o Alembric.

Para este trabajo, se ha optado por utilizar Blender como *software* base para gestionar modelos o generar *assets*. Blender ofrece un flujo de trabajo bastante ágil, tanto para modelar como para la creación de texturas. Entre otras ventajas de utilizar Blender encontramos el hecho de que es un *software* completamente libre, con multitud de documentación online y *addons*[38] que facilitan procesos en la creación de nuestros modelos 3D. Es importante comentar que Blender también tiene un motor en tiempo real. Es el caso de Evee un motor que procesa por medio de GPU y que da resultados de buena calidad, pero que tiene carencias cuando se quiere obtener un resultado fotorrealista.

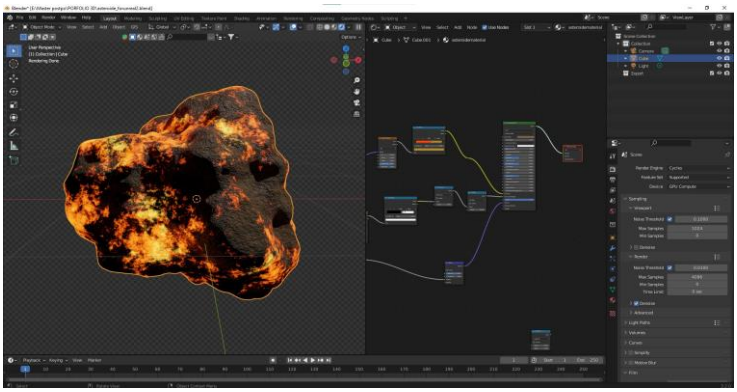

Figura 11. Blender interface

Para los procesos de *postproducción* tanto de los VFX como del 3D en general, se utilizan otros programas en los cuales se realiza la composición, corrección de color y otros procesos que contribuyen en el acabado final. Algunos de estos programas son Nuke[39] o DaVinci Resolve[40]. Para este trabajo se ha elegido utilizar DaVinci debido a que tiene unas herramientas muy potentes e intuitivas para editar y monitorear el color.

[38] Addons. Complementos para Blender como plugins externos o scripts.
[39] Nuke. Software de composición digital desarrollado por The Foundry.
[40] DaVinci Resolve. Software libre para edición de video y corrección de color, desarrollado por Blackmagic.

3.2.- Pipeline

Antes de generar cualquier escena o efecto visual, se ha diseñado un flujo de trabajo orientado a la producción de efectos visuales 3D, con el fin de organizar y planear los diferentes procesos que se van a realizar en la producción de los efectos visuales detallados después de este apartado. El pipeline ilustrado en la figura 12, se ha utilizado en los tres proyectos que componen este TFM.

Para la realización de los diferentes efectos visuales, en primer lugar, se han ideado diferentes *sets* donde recrear dichos efectos. Para ello, se han buscado diferentes modelos libres con los cuales se han configurado los diferentes *sets* y *enviroments*. Después se ha diseñado una iluminación y se ha creado el tiro de cámara inicial. Se ha continuado generando las animaciones que se necesitaban, para después realizar los diferentes efectos visuales propuestos. Después se ha efectuado una fase de postproducción donde se han añadido efectos de *postprocesado*, se ha hecho la composición y una corrección de color sencilla para poder obtener un resultado final.

Cabe comentar que en un *pipeline* profesional las fases de iluminación y cámara se realizan después de la producción de VFX. En este caso se ha cambiado debido a que las diferentes animaciones están realizadas por una persona y no por un equipo. El adelantamiento de estas dos fases ha resultado producir un flujo de trabajo más ágil en esta situación.

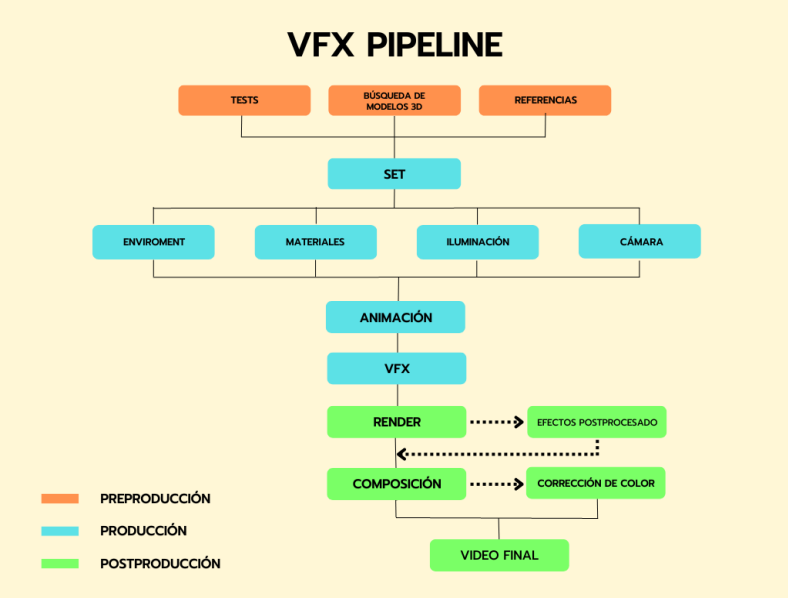

Figura 12. Flujo de trabajo utilizado en este TFM.

3.3.- Proyecto 1. Nieve y niebla

3.3.1.- Set

En este primer proyecto se plantea la creación de niebla y nieve animada mediante un sistema de partículas. Para ello, se ha creado un ambiente polar. El set es simple pero óptimo para realizar una simulación ambiental (Figura 13).

Figura 13. Frame del resultado final del proyecto 1

En primer lugar, se ha creado un océano a gran escala, Unreal Engine tiene herramientas y clases predefinidas para la creación de diferentes océanos. Para este set se ha creado un océano con la herramienta "*water body ocean*". Esta herramienta permite la creación de un sistema de agua basado en *splines*[41] que consta de dos elementos clave: una superficie de agua editable y un material para esta superficie.

Al utilizar esta herramienta se genera una malla de agua editable y también un dominio dentro del cual se ubica el agua. En este caso para editar la malla de agua hemos editado diferentes parámetros como el *wavelength*[42] o la amplitud (Figura 14).

[41] Spline. Curvas compuestas por polimios utilizadas para crear puntos de control y realizar interpolaciones
[42] Wavelenght. En Unreal Engine es un parámetro para controlar las ondulaciones de la malla en el sistema de agua.

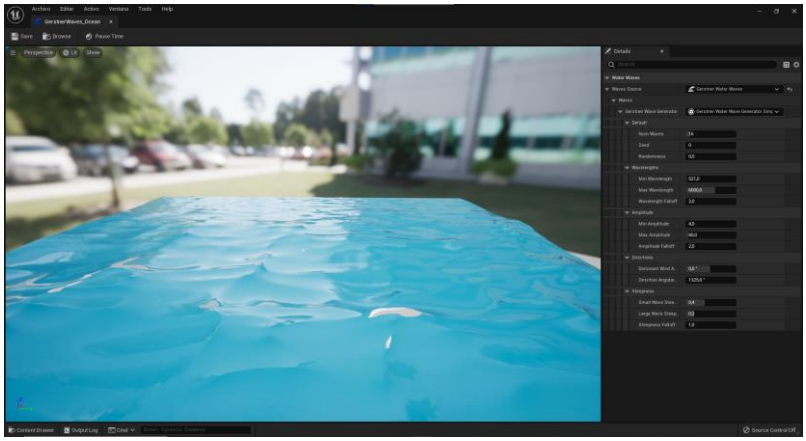

Figura 14. Configuración de las olas

Una vez importada la clase en la escena, hemos cambiado el material predefinido por otro de mayor calidad y con otras tonalidades más oscuras, el cual se incluye también en el contenido del motor. Con el objetivo de simular un océano más propio de un clima polar, a este material le hemos añadido caústicas [43] para obtener unos resultados más realistas, obteniendo los resultados que se muestran en la figura 15.

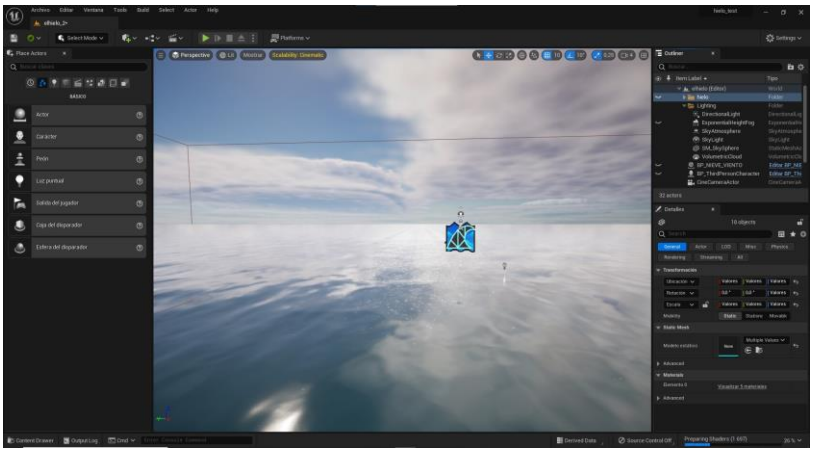

Figura 15. Océano creado para el proyecto 1

Para completar el set, se han descargados diferentes modelos de rocas con hielo de la librería de Quixel megascam, y posteriormente se han distribuido con el fin de recrear una costa polar (Figura 16). También hemos activado la opción de Nanite en la geometría de dichos modelos con el fin de optimizar el rendimiento y obtener

[43] Cáuticas. Luces que se forman en una superfície donde la luz se refleja.

mejores resultados. Por último, se ha adecuado la luz a la escena mediante la herramienta *Enviroment light mixer*, que permite gestionar tanto la iluminación como la atmosfera o las nubes de la escena (ver anexo 2).

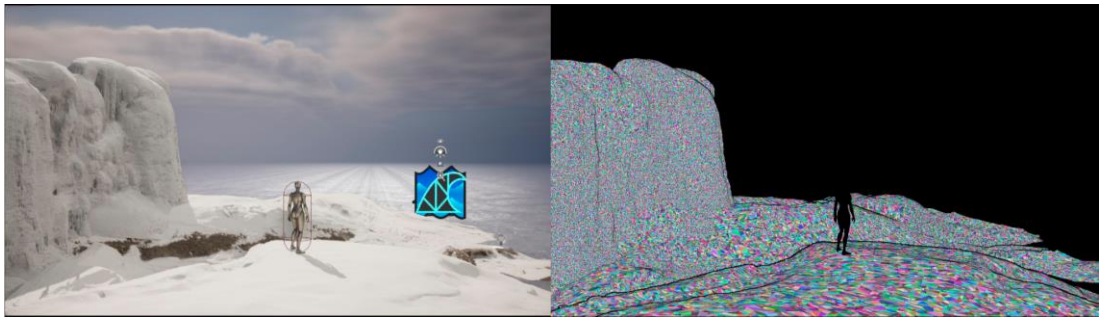

Figura 16. Set y vista *Nanite* de la geometría.

3.3.2.- Efectos visuales

Para realizar los efectos visuales de esta escena se han creado dos sistemas de partículas. Esto se ha realizado con el fin de sustituir las partículas de cada sistema por nieve o humo para simular niebla que se mueve. Para ello, se ha utilizado la herramienta Niagara.

Para la simulación de la nieve, se ha creado un sistema de partículas a partir de la plantilla "*hanging particles*", y esta crea en la interfaz de Niagara un sistema con una configuración básica. Los sistemas que se crean dentro de Niagara funcionan de forma modular, donde cada módulo representa un atributo con sus respectivos parámetros que afectan al comportamiento del sistema de partículas (Figura 17).

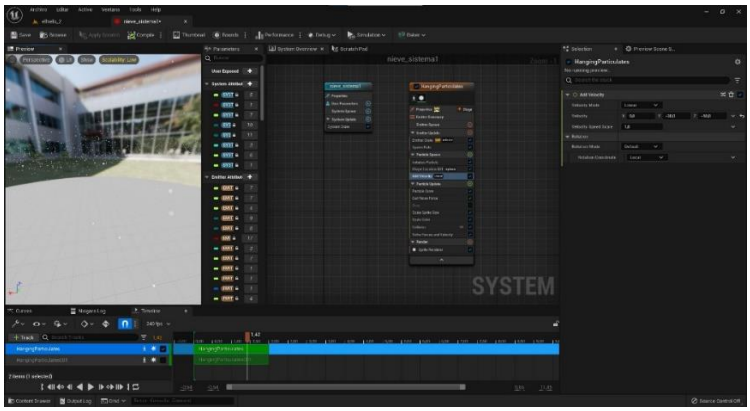

Figura 17. Sistema de partículas básico en Niagara.

En primer lugar, se ha cambiado la forma en la que se computa la simulación, ya que por defecto viene computado por CPU[44]. Para optimizar la simulación se ha cambiado a la computación del sistema por GPU. Después se ha aumentado el *spawn rate*, parámetro que nos permite especificar el número de partículas del sistema. Por otro lado, también se ha reconfigurado el *lifetime* de estas partículas, parámetro con el cual se puede especificar un tiempo de vida máximo y mínimo para las partículas, y también se ha modificado el tamaño de la celda donde se emiten las partículas. Para continuar, se ha añadido el atributo de velocidad, de esta manera le podemos dar valores de velocidad a las partículas y direccionarlas. Para este efecto se han puesto en negativo los valores de velocidad en Z y en Y. En Niagara los atributos se añaden en forma de módulos, luego podemos abrir estos módulos y acceder a los parámetros

También se ha modificado el atributo *Curl Noise* con el fin de crear una animación más realista. Este atributo nos genera una turbulencia en la animación de las partículas, la cual podemos ajustar para conseguir el efecto de que las partículas están siendo movidas por un viento.

Figura 18. Sistema de partículas en el set

Para la recreación de la niebla, se ha creado un segundo sistema de partículas. En este caso también se ha empezado con la plantilla "*hanging particles*", ya que nos genera partículas en una localización aleatoria (dentro del *grid*) que nacen y luego mueren. Para crear la niebla se ha aumentado el tamaño y el número de las

[44] CPU. Central Processing Unit. Unidad de procesamiento del ordenador.

partículas, y se han substituido estas por una textura 2D de humo. En el módulo de render, ubicado al final de nodo que se puede ver en la figura 19, podemos cambiar el material de salida del sistema de partículas. Por último, en este módulo, también se ha ajustado los UVs para que la textura se vea a un tamaño correcto.

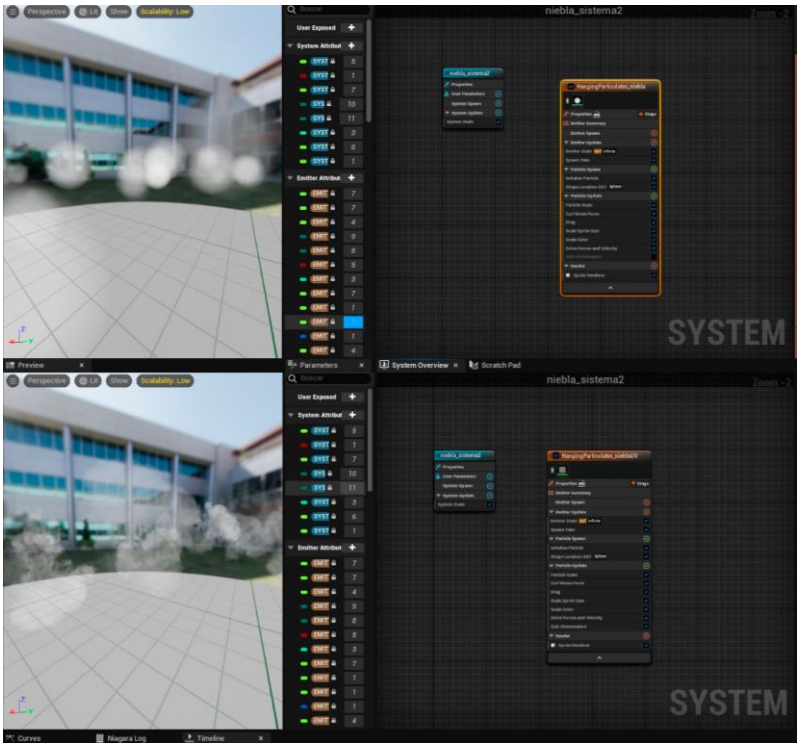

Figura 19. Sistema de partículas emitiendo una textura

Para poder juntar los dos efectos, se ha recurrido a la creación de otro sistema de Niagara, en el cual se han copiado los nodos generados en los sistemas anteriores, realizando modificaciones en tamaño y posición para que tengan coherencia entre sí. Por último, se ha creado un *spline* para que marque la dirección de la niebla y la nieve, de forma que la nieve y la niebla vaya generándose en línea recta hacia la cámara. Para poder hacer esto se ha creado un *blueprint*[45]. En este *blueprint* se ha importado el archivo generado con Niagara con nuestros sistemas de partículas y se ha creado un *spline* recto, el cual se ha conectado con los sistemas de partículas (Figura 20).

[45] Blueprint. Sistema de scripting visual con el que se ejecutan secuencias de comandos mediante una interfaz nodal.

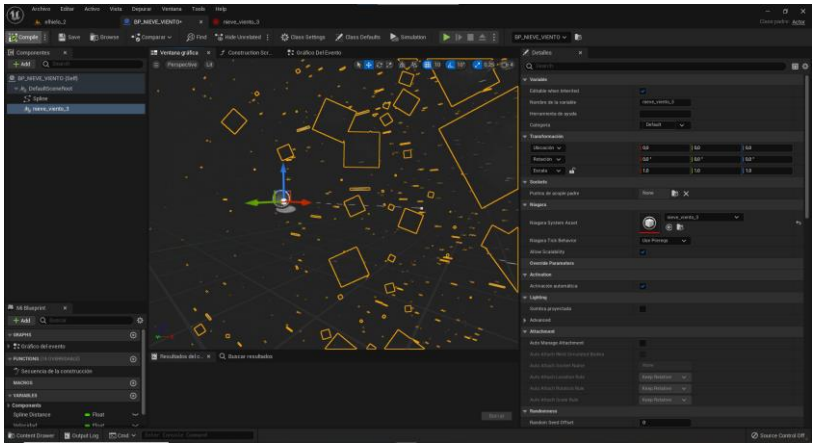

Figura 20. Blueprint con los sistemas de partículas y el spline

3.3.3.- Render

Para acabar nuestra escena se ha creado un *sequence level,* que es la herramienta con la que gestionamos el *timeline.* En esta se ha importado la cámara y se ha animado creando *keyframes* para resolver la cinemática.

Finalmente la escena se ha exportado en una secuencia del formato EXR[46] para composición en una resolución de 1920x1080. Al *pipeline* del render, se le ha añadido *antialising* de 128 *samples* y se han obtenido los resultados reflejados en la figura 21.

Figura 21. Render final proyecto 1

[46] .EXR. Formato para imágenes de alto rango dinámico en el que se pueden incluir diferentes pases de render.

3.3.4.- Composición y corrección de color

Una vez exportada la secuencia de EXR mencionada en el apartado anterior se ha realizado el proceso de composición y corrección de color en el software DaVinci Resolve.

Para obtener los resultados que se muestran en la figura 22, en primer lugar, se ha importado la secuencia de EXR en DaVinci. Después se ha creado un nodo para llevar a su sitio los blanco y negros utilizando las *color wheels*[47] y se le ha dado algo de contraste. Se ha continuado creando otro nodo donde se ha bajado la temperatura, y se han acentuado un poco los tonos azules. También se ha añadido un nodo de *glow* para exagerar el brillo de la imagen y, en último lugar, se ha creado un nodo de *Aperture Diffraction* el cual resalta los estallidos de luz, mejorando el efecto producido por el *glow*.

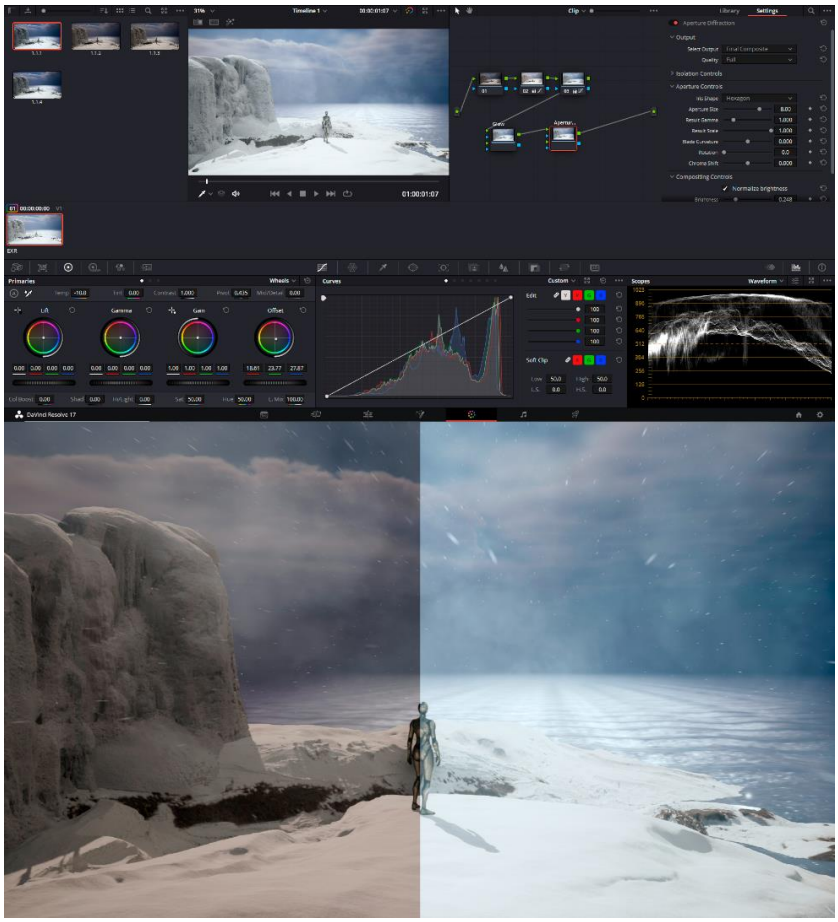

Figura 22. Corrección de color del proyecto 2 y comparación.

[47] Color wheels. Ruedas de color. Herramienta de DaVinci Resolve para editar parámetros de color.

3.4.- Proyecto 2. Simulación de humo

3.4.1.- Set

En este segundo proyecto se tratan simulaciones de explosiones, fluidos (humo) y sistemas de partículas. Para este ejemplo se ha configurado un set ambientado en un asentamiento en un planeta rocoso similar a marte, donde una nave intenta realizar un aterrizaje forzoso, pero sin embargo acaba explotando (Figura 23).

Figura 23. Frame del resultado final del proyecto 2.

También se ha recreado un terreno donde ubicar el asentamiento. Para la construcción de este set se ha utilizado modelos de Quixel Megascam para las rocas, y modelos de Kitbash para la nave espacial y las estructuras (figura 24).

Figura 24. Set e iluminación del proyecto 2

Una vez construido el set, se ha creado una iluminación basada en tonos amarillos y naranjas con el fin de emular una atmosfera propia de un planeta similar a Marte. Para ello se han cambiado los colores del *Sky Atmosphere* y del *Skylight* y se ha aplicado niebla volumétrica a la escena. En el anexo 3 se puede consultar una comparativa del set con y sin iluminación, y las diferentes vistas de Nanite.

3.4.2.- Efectos visuales

Para este proyecto se han realizado diferentes efectos. En concreto se han utilizado *VFX*, para los propulsores de la nave, el humo que emerge de la nave y su explosión.

En primer lugar, los propulsores de la nave se han resuelto con un sistema de partículas. Para ello, se ha creado un archivo de Niagara utilizando la plantilla *Fountain*, la cual simula una fuente de partículas. Partiendo desde esta plantilla, se ha cambiado el parámetro *Shape Location*, parámetro donde se puede definir la geometría desde la cual se emiten las partículas en este caso se ha elegido una esfera. También se ha aumentado el número de partículas a 4000 y se ha aleatorizado su tamaño. Después se le ha añadido un atributo de velocidad en el eje X para que las partículas se muevan más rápido. También se ha utilizado el atributo *scale color*, el cual escala el color de las partículas según su vida útil. En este atributo mediante una curva se ha modificado la opacidad de las partículas. En último lugar, se ha creado un material emisivo azul para cargarlo en el sistema de partículas (Figura 25).

Figura 25. Sistema de partículas para el propulsor

Como la nave tiene que estar animada, hay que emparentarle el sistema de partículas. Para esto se ha tenido que crear un *blueprint por* cada propulsor en escena, donde se ha cargado la geometría de los propulsores y los archivos generados con Niagara (Figura 26). Después, se ha duplicado el sistema de partículas y se han colocado en los propulsores. Una vez situado se han emparentado ambos sistemas a la geometría del propulsor. Finalmente, estos *blueprints*, que incluyen dentro los propulsores y los sistemas de partículas se han emparentado al cuerpo de la nave, el cual se ha animado.

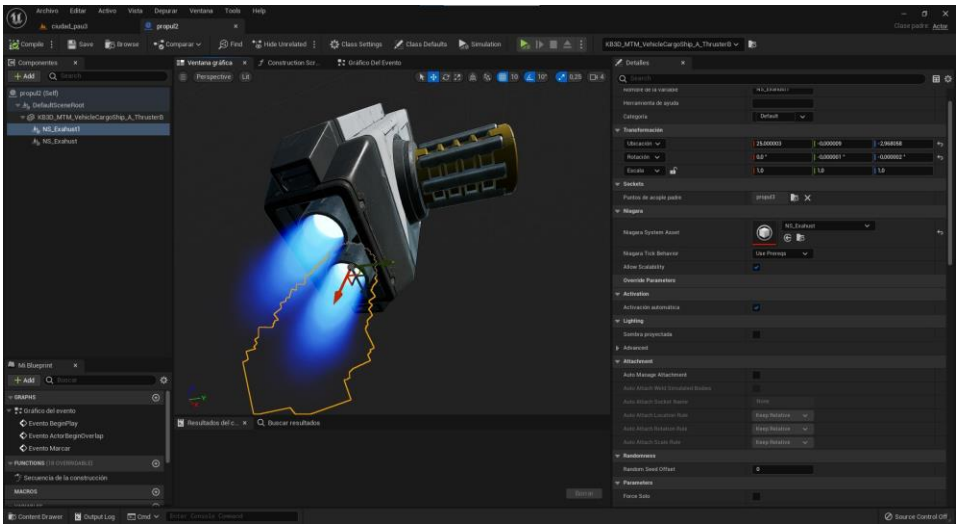

Figura 26. Blueprint con el sistema de partículas

Para crear el humo, se han añadido al motor Niagara Fluids, contenido adicional del motor que contiene *shaders* y plantillas para la creación de fluidos con la herramienta Niagara.

Para crear el humo que emite la nave se ha empezado a partir de la plantilla *Grid 3D gas simple particle source*, la cual contiene una simulación de gas en 3D que proviene de un emisor de partículas. El archivo de Niagara generado contiene tres nodos para controlar el sistema (Figura 27). El primero de ellos, de color azul, sirve para controlar parámetros de usuario, los cuales pueden ser controlados también desde el *timenline*. Los otros nodos son los que controlan la simulación de gas. Por un lado, tenemos el nodo *Particle Source Emitter*, nodo con el cual controlamos el sistema de partículas origen, en este se han incluido atributos para modificar la simulación base, también

se han modificado algunos atributos que venían por defecto. También encontramos el nodo *Grid 3D Gas Controls Emitter*, en el cual se añade el fuego (temperatura) y humo (densidad). Este nodo también contiene *shaders* para las simulaciones, de esta forma también se puede realizar un sistema de partículas propio y luego referenciarlo a este nodo.

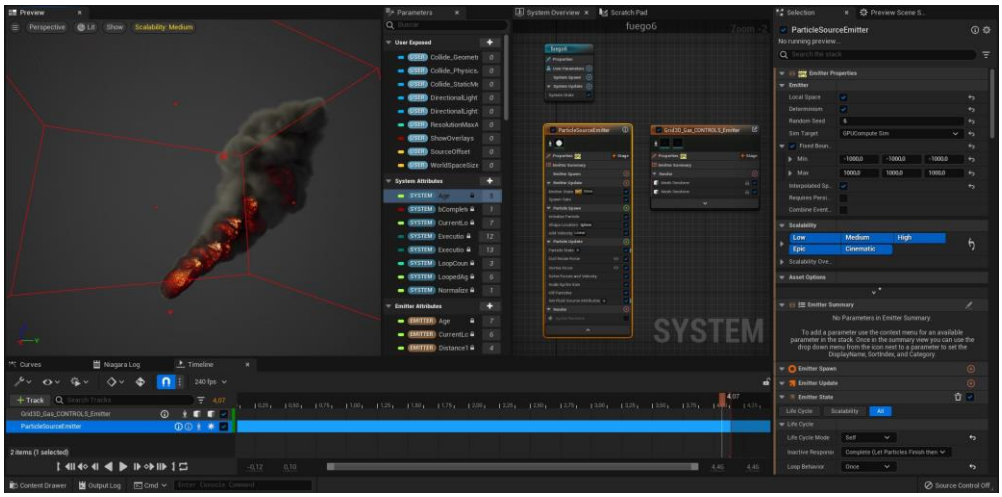

Figura 27. Creación de humo en Niagara.

Para la creación de este efecto, en primer lugar, se ha modificado el atributo de velocidad para direccionar el humo, después se ha modificado el *curl noise force* y se ha añadido el atributo *vortex force* para crear más turbulencia. Finalmente, se ha tenido que escalar para que tuviera coherencia con la escena, y para esto se han tenido que modificar las celdas del dominio, la geometría emisora y el número de partículas del sistema, el cual finalmente se ha elevado a 20.000.

Para hacer la explosión, también se ha partido de la misma plantilla que para el humo, pero se ha substituido el módulo de *spawn*, por el módulo uno denominado *spawn burst instantaneous*, el cual hace que el emisor genere ráfagas. Después, se ha cambiado la geometría emisora por un *torus* (toroide), con el objetivo de que las partículas se expandan con esta forma (Figura 28).

Una vez creado este sistema se ha jugado con los valores de densidad y temperatura, también se ha modificado el *curl noise* a fin de crear turbulencias en la emisión del humo.

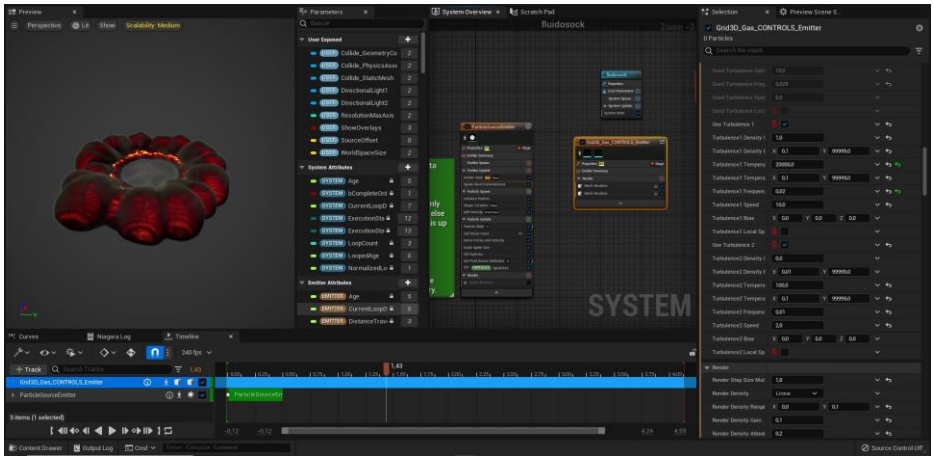

Figura 28. Creación de la explosión.

Una vez acabado el sistema de Niagara, se ha importado en escena y se ha situado en la localización donde se pretende explotar la nave. Para que la explosión suceda en un momento concreto de la animación, se ha tenido que desactivar el modo bucle, activado por defecto, y posteriormente se ha importado en el *timeline*, también se ha importado el parámetro (del sistema de Niagara) *Spawn Life Cycle*, en el cual nos permite marcar en el *timeline* cuando empieza y termina la simulación de la explosión.

En último lugar, con el fin de hacer la explosión más creíble, se ha creado un efecto de destrucción en la nave, para ello se ha fracturado la geometría de la nave. Esto se ha hecho mediante el modo fractura, donde podemos romper una geometría de diferentes formas (Figura 29).

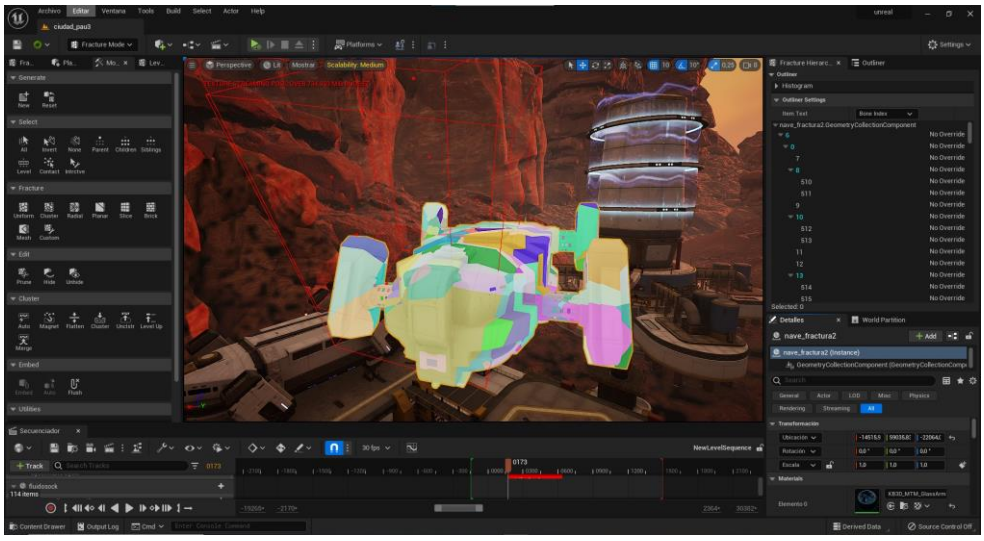

Figura 29. Modo fractura en Unreal Engine.

En este caso se ha elegido fracturar la geometría de manera uniforme. En este modo se permite la opción de fracturar con niveles, teniendo cada nivel un tamaño de fractura diferente, para crear variedad en las piezas que se fracturan. Para este efecto se han hecho tres niveles de fractura suficientes para que haya variación en el tamaño de las piezas.

Para crear la animación de la fractura, se han activado las físicas y colisiones en el objeto fracturado y se le ha establecido una masa (en kg). Después, lo que se ha hecho es colisionar la nave contra un objeto invisible en la localización exacta donde ocurre la explosión. Una vez realizada la simulación, se ha realizado un proceso de *bake*, necesario para liberar la *caché* y poder importar la fractura en el *timeline*, donde se ha coordinado en tiempo con la explosión (Figura 30).

Figura 30. Destrucción de la nave.

3.4.3.- Render

Una vez acabados los diferentes efectos se ha *trackeado* la cámara a la nave y se han probado diferentes movimientos de cámara hasta conseguir elaborar tres planos, lo cuales conforman la cinemática que se puede consultar en los anexos.

Este proyecto se ha exportado en secuencia de formato *.EXR* para composición, con una resolución de 1920 x1080. También se han aplicado efectos de postprocesado, *en este caso bloom* y *motion blur*. (Figuras 31 y 32).

Figura 31. Frame del proyecto 2

Figura 32. Frame del proyecto 2

3.4.4.- Composición y corrección de color

Para realizar la composición y corrección de color en este proyecto se han importado las tres secuencias con formato .EXR en DaVinci Resolve. Después se ha editado el corte entre los planos y se ha continuado con la corrección de color.

Para hacer las tres escenas se ha operado de forma similar, aquí también se ha creado un nodo en el cual se han llevado los blancos y negros a sus valores reales,

esto se ha hecho mediante el uso del histograma y los medidores de forma de onda que utiliza DaVinci Resolve. Después se ha creado un nodo en el cual se han modificado los tonos rojos, y en último lugar, se ha creado un nodo *aperture diffraction* con el cual se han intensificado los brillos de los materiales emisivos de la escena (Figuras 33 y 34).

Figura 33. Corrección de color del proyecto 2

Figura 34. Comparación de la corrección de color del proyecto 2

3.5.- Proyecto 3. Simulación de fluidos

3.5.1.- Set

En este proyecto se plantea una simulación de fluidos con colisiones, para ello se ha montado un set que consta de una explanada rocosa donde hay un tanque con plasma del cual emerge una nave (Figura 35). Las diferentes rocas son modelos de Quixel Megascam, las cuales se han distribuido por el terreno. Por otro lado, se ha descargado un modelo 3D gratuito de la famosa nave *X-Wing* de *The Star Wars* y se ha colocado en un tanque para el líquido creado a partir de un cubo (Figura 36).

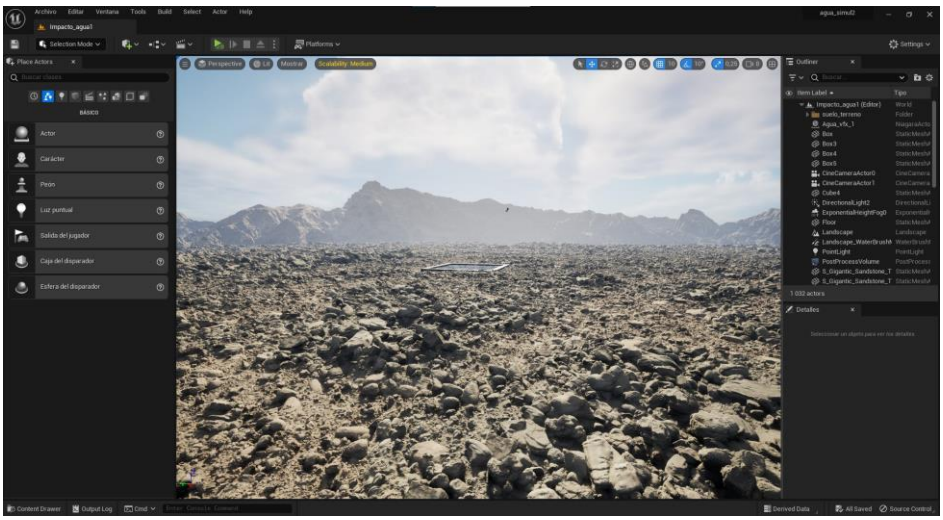

Figura 35. Set del proyecto 3

Figura 36. Frame del resultado final del proyecto 3.

Para la creación de este set también se ha creado niebla volumétrica, con el objetivo de crear un ambiente más realista, también se ha diseñado una iluminación de acorde con el set. Para poder optimizar toda la geometría de las rocas, también se ha utilizado Nanite, obteniendo los resultados que se pueden observar en la figura 37. En el anexo 4 se puede consultar una comparativa del set con y sin iluminación, además de las diferentes vistas Nanite.

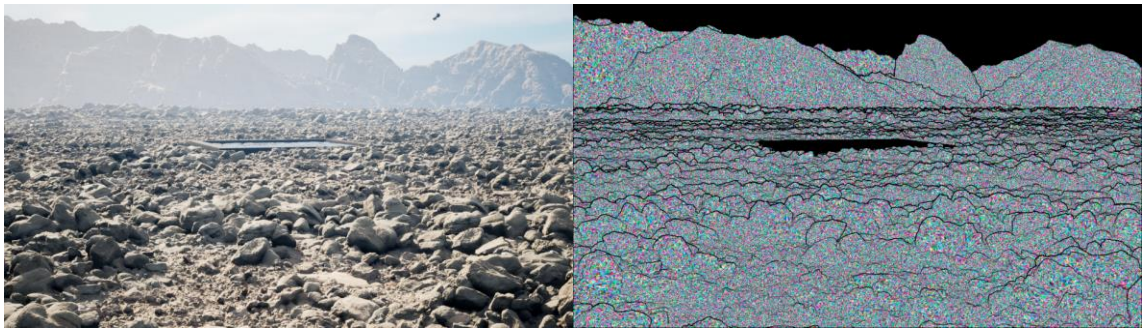

Figura 37. Set y vista *Nanite* del proyecto 3

3.5.2.- Efectos visuales

En este ejercicio se ha planteado la creación de un líquido viscoso que colisiona con la nave la cual emerge de este mismo líquido. Para ello, se ha utilizado la herramienta Niagara y el contenido de *Niagara Fluids*, como en el ejercicio anterior.

A partir de la plantilla *Grid 3D FLIP Pool*, se ha creado un sistema de líquido, el cual en un principio se ha escalado para que tenga coherencia con la escena. Después se han activado las colisiones por medio de la utilización del *tag*[48] *collider* en el sistema de Niagara y en la geometría que va a colisionar con el líquido, que en este caso es la nave. Una vez hecho esto, el programa entiende que estas dos geometrías tienen que interactuar entre si y ya se obtienen colisiones en tiempo real.

También se han modificado diferentes parámetros del sistema de Niagara para conseguir cierta viscosidad en el líquido simulado. Para esto se ha modificado el *parámetro Pressure Interaction*, el cual se ha reducido a 70. Por otro lado, también se ha cambiado el número de celdas del sistema, y las partículas que existen por cada celda, con el fin de aumentar interacciones (Figura 38).

[48] Tag. Etiqueta conceptual y jerárquica a través de nombres creados por el usuario.

En último lugar, se ha realizado la animación de la nave, la cual ya interactúa con el sistema líquido, mediante el uso de dos *keyframes*.

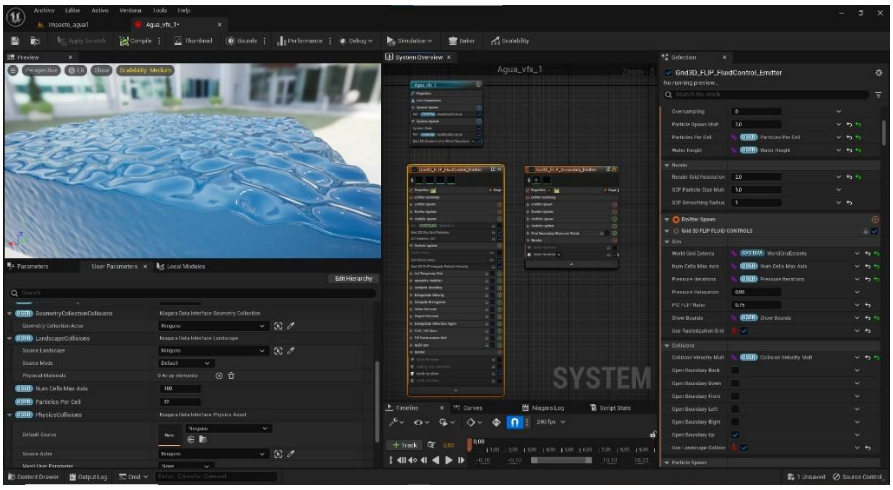

Figura 38. Sistema de líquidos en Niagara.

3.5.3.- Render

Para este proyecto se han creado varios planos, esto se ha hecho mediante la creación de 2 cámaras ambas mapeadas a la nave. Después de diseñar la cinemática y animar ambas cámaras se han obtenido los siguientes resultados los cuales se pueden consultar en la figura 39, y en el video de la animación final disponible en los anexos.

Figura 39. Animación de las cámaras del proyecto 3.

3.5.4.- Composición y corrección de color

Para realizar la fase de composición y corrección de color, se han importado a DaVinci Resolve las dos secuencias con formato EXR, después estas se han importado al *timeline* y se ha editado el cambio de plano.

En este proyecto se ha hecho una corrección de color muy simple pero efectiva. Se ha creado un nodo para poder llevar los blancos a un nivel más coherente, y después se ha subido también la gamma. Después se ha creado un poco de contraste y se ha reducido la temperatura (Figuras 40 y 41).

Figura 40. Corrección de color del proyecto 3

Figura 41. Comparación de la corrección de color del proyecto 3

4.- Conclusiones

La creación de efectos visuales en Unreal Engine ha terminado suponiendo una labor bastante costosa, pero tras mucho esfuerzo se han conseguido cumplir los diferentes objetivos propuestos.

El objetivo principal de este TFM era estudiar y elaborar efectos visuales en 3D mediante sistemas partículas y fluidos, utilizando *software* 3D y herramientas específicas para la creación de estos. Durante la parte de contextualización se han estudiado estos sistemas y posteriormente se han solventado diferentes efectos propuestos los cuales pueden consultarse en los anexos. Por otro lado, encontramos que se han utilizado nuevas tecnologías como la herramienta Niagara o los *Nanite* en Unreal Engine, nuevas herramientas y tecnologías 3D, y también se han desarrollado habilidades en el manejo de este *software* cumpliendo con los objetivos secundarios propuestos.

En el proceso de creación de los diferentes efectos visuales se han tenido que solucionar ciertas dificultades las cuales se detallan a continuación. En el primer proyecto se ha tenido problemas para inicializar los dos sistemas de partículas de forma que tuvieran coherencia entre sí y siguieran la misma dirección. Para esto al final se ha utilizado un *spline*, como se indica en el apartado del proyecto 1. Por otro lado, en el segundo proyecto se han tenido algunos problemas de escala en los efectos de humo, también en la generación de fuego. Los problemas de escala, han provocado que se pierda calidad en los efectos de la explosión y el humo. También, en este proyecto se ha llegado al límite de partículas que soportaba el ordenador, y esto ha provocado algunas pérdidas de calidad en el renderizado final. En el proyecto 3 también se ha alcanzado el límite de partículas y esto ha supuesto tener que reducir el número de interacciones en las colisiones de estas.

En conclusión, en este trabajo se ha podido experimentar de una forma práctica la generación de efectos visuales, siendo esta una forma directa de aprender y probar en escena diferentes efectos visuales. En el proceso de creación tanto de los *sets* como de los efectos visuales se han adquirido conocimientos creado una base en el manejo de Unreal Engine y en el uso de sistemas de partículas para generar VFX, con la cual afrontar futuros proyectos.

5.- Referencias Bibliográficas

Bridson, R. (2018). *Fluid Simulation for Computer Graphics*. CRC Press.

Byrne, B. (2009). *The visual effects arsenal: VFX solutions for the independent filmmaker*. Focal Press.

Charles Finance, & Zwerman, S. (2009). *The visual effects producer: Understanding the art and business of VFX*. Focal Press.

Contreras, J. O. (2013). *Efficient algorithms for the realistic simulation of fluids*. Universitat Politècnica de Catalunya.

Dinur, E. (2017). *The filmmaker's guide to visual effects: The art and techniques of VFX for directors, producers, editors and cinematographers*. Routledge.

García, M. A. D. (2010). *Análisis de los avances digitales para el desarrollo e integración de la animación tradicional y la animación generada por ordenador en películas históricas*. Universidad Politécnica de Valencia.

Haines, E., Hoffman, N., & Akenine-Möller, T. (2018). *Real-Time Rendering, Fourth Edition* (4th ed.). A K PETERS.

Openwebinars.net. (13 junio 2019). *Origen, evolución y versiones de Unity*. https://openwebinars.net/blog/origen-evolucion-versiones-unity/

Reeves, W. T. (1983). *Particle systems---a technique for modeling a class of fuzzy objects. Proceedings of the 10th annual conference on Computer graphics and interactive techniques - SIGGRAPH '83*.

RenderMan at 30: A visual history. (27 noviembre 2018). VFX Voice Magazine; VFX Voice. https://www.vfxvoice.com/renderman-at-30-a-visual-history/

Sánchez, S. M. (2018). *Evolución de los efectos visuales en la historia del cine y su influencia sobre la industria del video musical*. Universidad Complutense de Madrid.

Shirley, P., Ashikhmin, M., & Marschner, S. (2009). *Fundamentals of computer graphics* (3a ed.). CRC Press.

Seymour, M. (15 septiembre 2011). *The science of fluid sims*. Fxguide. https://www.fxguide.com/fxfeatured/the-science-of-fluid-sims/

Unrealengine.com.(13 octubre 2022) *NANITE FOR EDUCATORS AND STUDENTS*. https://cdn2.unrealengine.com/nanite-for-educators-and-students-2-b01ced77f058.pdf

Wells, P. (2006). *Fundamentos de la animación.* Parramón

Wright, S. (2007). *Compositing visual effects: Essentials for the aspiring artist*. Focal Press.

Zwerman, S., & Okun, J. A. (Eds.). (2010). *The VES handbook of visual effects: Industry standard VFX practices and procedures*. Focal Press.

6.- Anexos

Anexo 1. Especificaciones del ordenador utilizado

Tabla de especificaciones	
Placa Base	Z390 M GAMING (Gigabyte)
CPU	Intel Core i7-9700F 3.00GHz
RAM	16 GB
GPU	NVIDIA GeForce RTX 2060 SUPER
Almacenamiento	HDD 1 GB SSD 1 GB
Sistema Operativo	Windows 10 Pro

Tabla 1. Características del equipo utilizado

Anexo 2. Proceso de creación del set del proyecto 1.

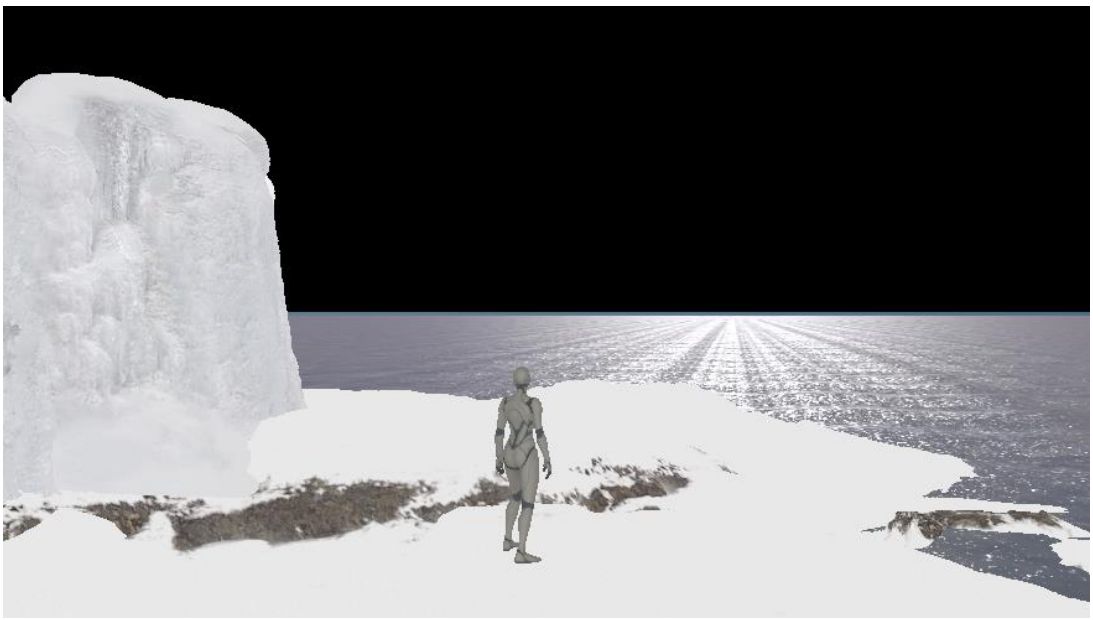

Figura 42. Set del proyecto 1 sin iluminación.

Figura 43. Set del proyecto 1 con iluminación.

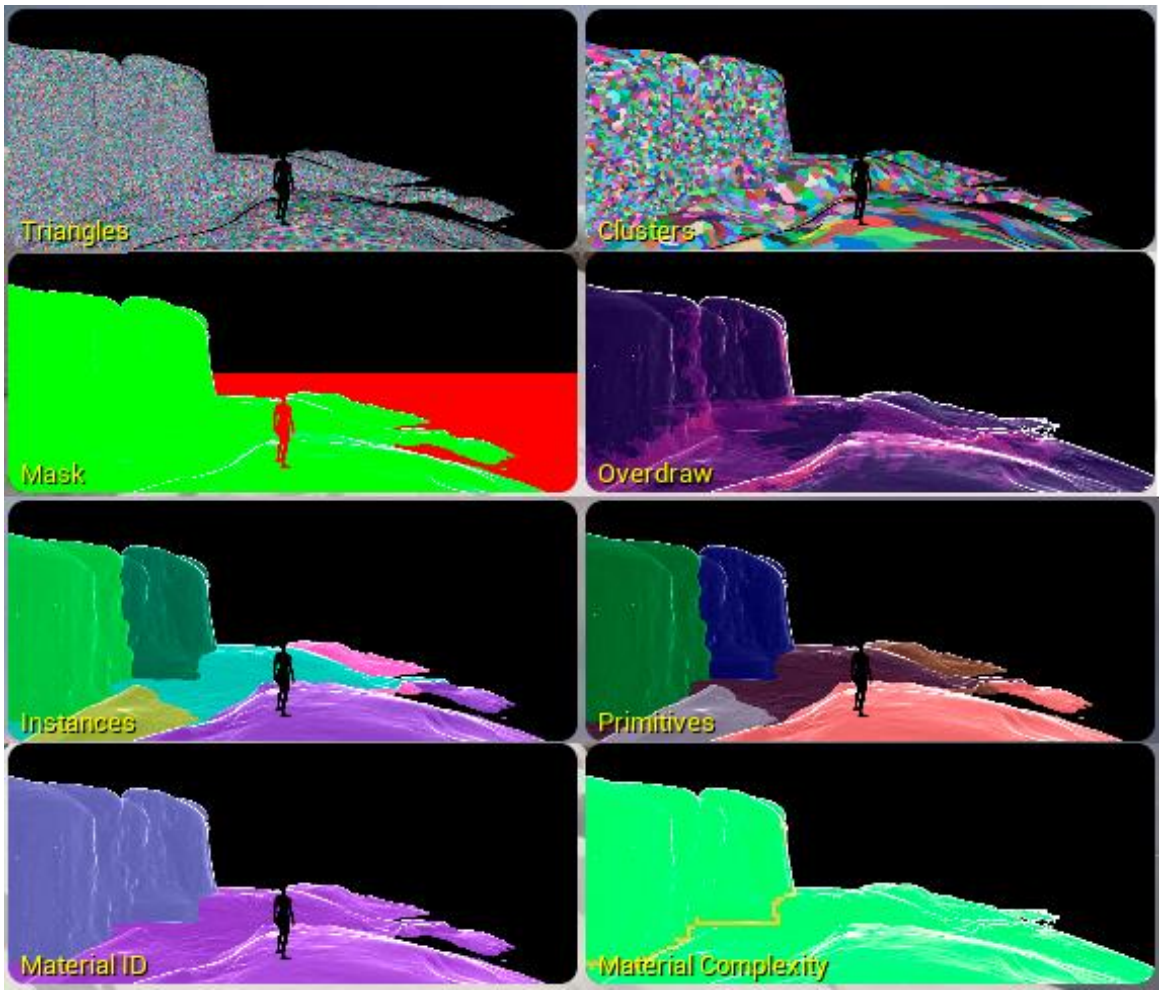

Figura 45. Vistas Nanite de la geometría del proyecto 1.

Anexo 3. Proceso de creación del set del proyecto 2

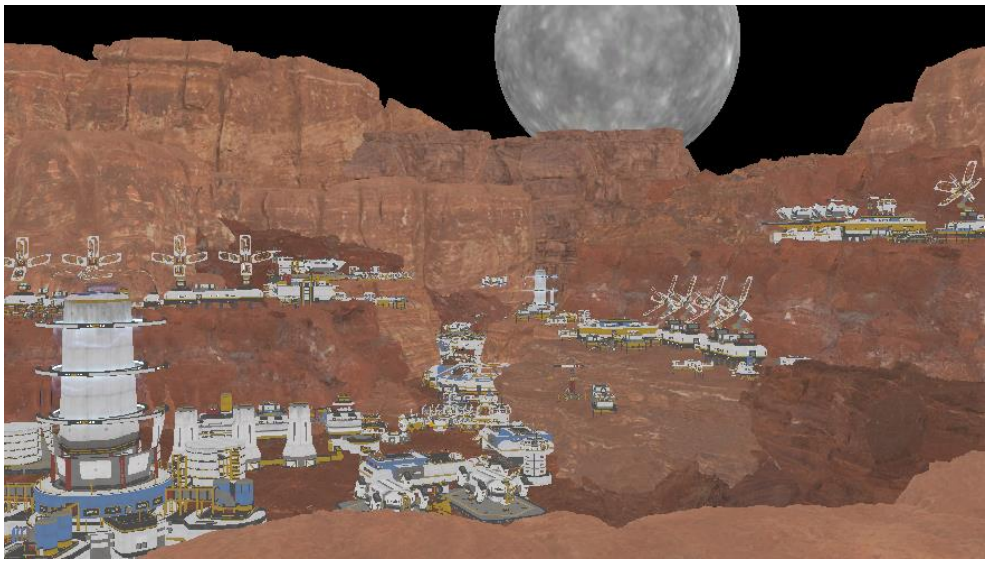

Figura 46. Set del proyecto 2 sin iluminación.

Figura 47. Set del proyecto 2 con iluminación.

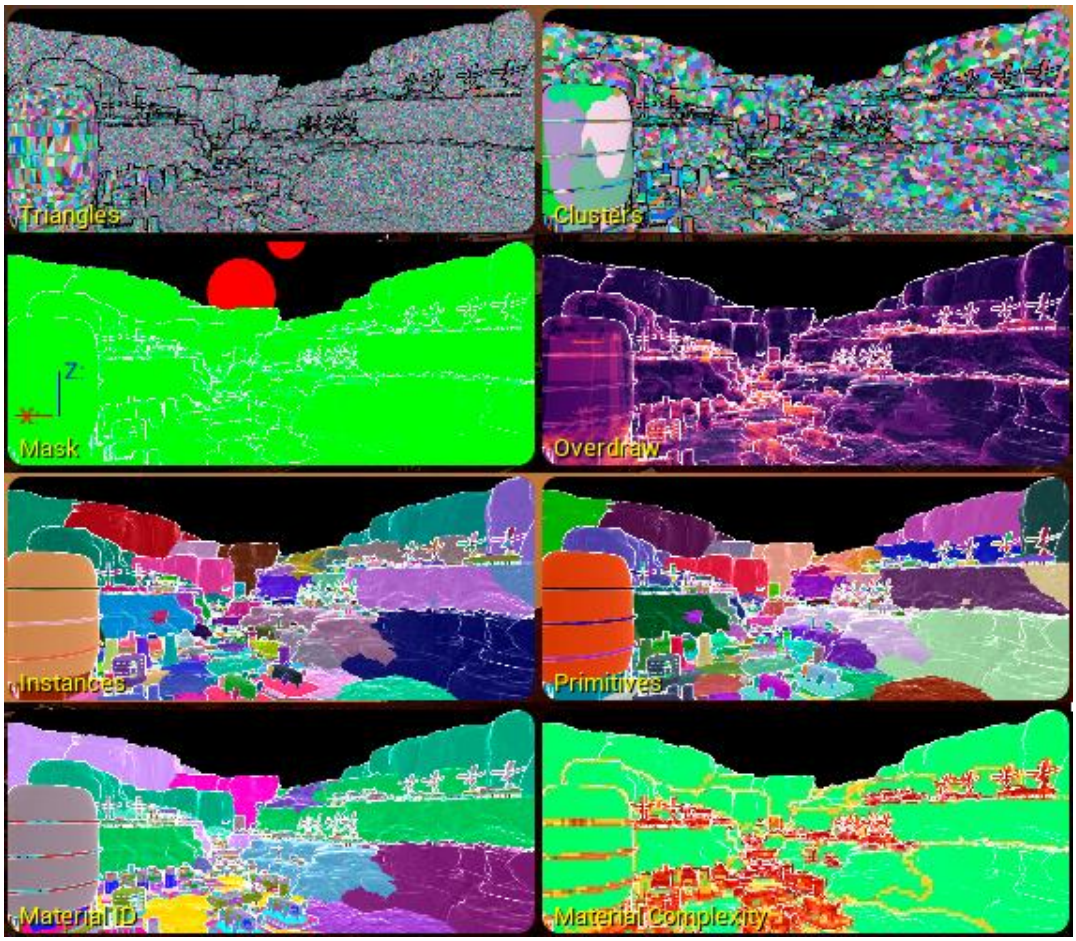

Figura 48. Vistas Nanite del proyecto 2.

Anexo 4. Proceso de creación del set del proyecto 3

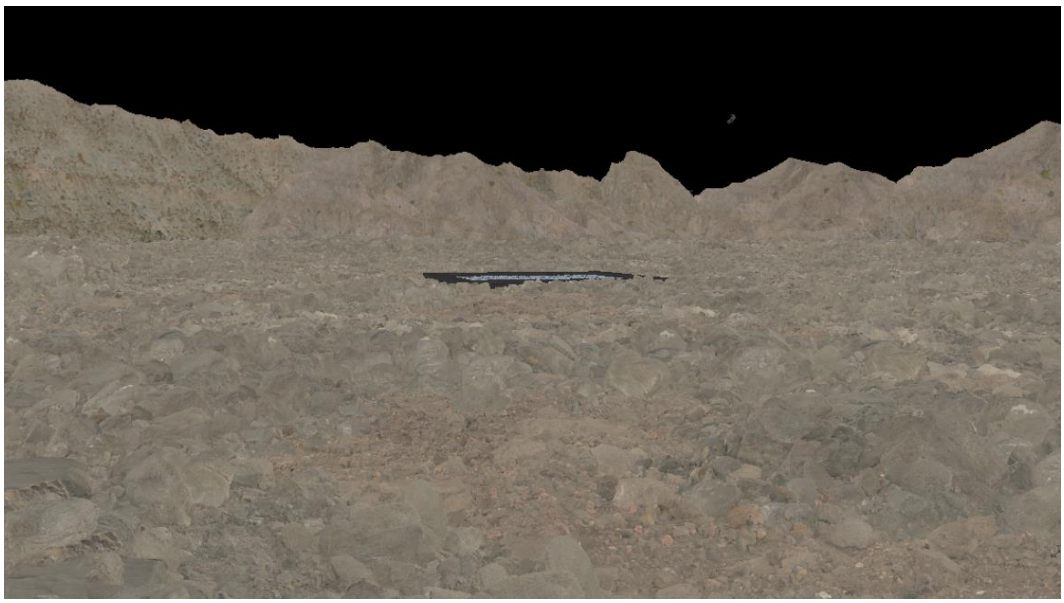

Figura 49. Set del proyecto 3 sin iluminación.

Figura 50. Set del proyecto 3 con iluminación.

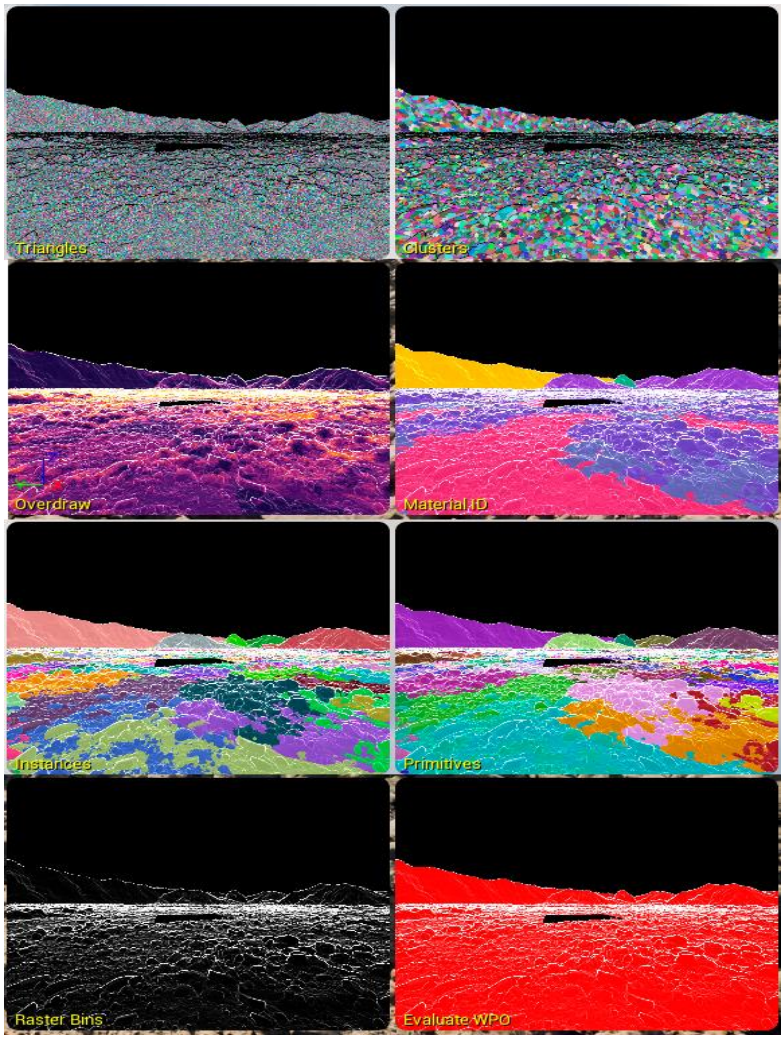

Figura 51. Vistas Nanite del proyecto 3.

Anexo 5. Videos de los proyectos.

Video 1: Proyecto 1. Nieve y niebla.

Video 2: Proyecto 2. Simulación de humo.

Video 3. Proyecto 3. Simulación de fluidos.