Instituto de Tecnología e Ingeniería

Introducción a los motores de videojuegos

Comisión N°1

Estudiantes: Ivan Enrique Estrada

Sistema de simulación de fluidos

Introducción a los motores de videojuegos **1**
Introducción **2**
Preguntas que me cuestiono 3
Objetivo 3
Propuesta 3
Resumen 3
Metodologías para la Simulación de Fluidos en Unity3D **4**
Shaders y sistema de VFX **4**
VFX Internos 4
VFX Externos 5
Recomendacion según el objetivo 5
Simulación mediante mallas 3D **5**
Recomendacion según el objetivo 6
Simulación de particulas **6**
¿Qué es SPH? 7
Müller y su implementación en Unity 7
Recomendacion según el objetivo 8
Comparación General de Métodos **8**
Respuesta a las Preguntas de Investigación **9**
Simulación en CPU vs GPU **9**
Conclusión Final **10**
Bibliografía **10**

Introducción

El motor de videojuegos elegido para este trabajo es Unity 3D, principalmente porque ya tengo experiencia previa con él, tanto en proyectos personales como universitarios. He trabajado en proyectos 2D y 3D, y creo que Unity 3D ofrece una combinación muy buena entre accesibilidad, documentación y potencia técnica. Por eso, me interesa seguir profundizando en sus capacidades más avanzadas, especialmente en la parte visual y física del motor.

El sistema que decidí investigar es el de simulación de fluidos, con el objetivo de entender cómo funciona dentro de Unity. Me parece un tema muy interesante porque los fluidos son un desafío técnico y también representan un punto clave para crear efectos y muchas veces

se vinculan directamente con la iluminación y el comportamiento de las partículas. La conexión entre ambos sistemas permite generar entornos más detallados y visuales.

Preguntas que me cuestiono

Mis principales preguntas de investigación son:

- ¿Qué tan complejos y eficientes pueden llegar a ser los fluidos simulados en Unity?
- ¿Cómo se pueden optimizar para su uso en tiempo real sin perder calidad visual?
- ¿De qué manera se relaciona la simulación de fluidos con el sistema de iluminación volumétrica del motor?
- ¿Qué técnicas o herramientas, como VFX o shaders permiten integrar ambos sistemas de forma normal?

Objetivo

El motivo de esta elección surge de mi interés por entender cómo los efectos visuales se construyen internamente, más allá del simple uso de herramientas. Quiero poder analizar la base técnica que hace posible que un humo interactúe con la luz, o que una simulación de líquido se vea realista en una escena iluminada dinámicamente. Además, creo que este tipo de integración puede ser aplicable a futuros proyectos personales, tanto para mejorar el rendimiento visual como para comprender cómo se comportan estos sistemas en conjunto.

Propuesta

El enfoque de trabajo que pienso seguir será teórico-práctico: por un lado, voy a investigar los fundamentos detrás de la simulación de fluidos (cómo se implementan, qué algoritmos se usan y cómo Unity los gestiona internamente mediante partículas y físicas) y explorar su relación con el sistema de iluminación.

La idea es poder llegar a una presentación de lo que es un sistema de fluidos, con una demostración técnica en un proyecto de unity, donde pueda mostrar las diferencias estéticas y de optimización en diferentes simuladores de fluidos. Donde se pueda ver y diferenciar a qué objetivo apunta cada uno.

Resumen

En resumen, el objetivo de este trabajo además de investigar los sistemas de fluidos, es entender la relación entre los fluidos simulados y la iluminación en Unity, explorando tanto los fundamentos técnicos como las aplicaciones prácticas dentro del motor. Todo el proceso estará orientado a poder presentar una demostración final que combine ambos sistemas, mostrando el resultado de la investigación.

Metodologías para la Simulación de Fluidos en Unity3D

En principio, quiero partir de la base de que la simulación de fluidos en Unity no existe como un sistema único “oficial”, sino que es una combinación de distintos componentes, mi trabajo tendrá tres enfoques principales.. El primero estará basado en shaders y efectos VFX, los cuales se centran exclusivamente en la representación visual del fluido, utilizando efectos gráficos sin recurrir a partículas físicas ni a mallas tridimensionales, con el objetivo de lograr resultados eficientes en rendimiento.
El segundo enfoque abordará simulación de partículas de forma realista , donde se buscará replicar comportamientos naturales de fluidos mediante sistemas de físicas aplicado a partículas, priorizando la precisión dinámica y el realismo del movimiento.
El tercer enfoque se centrará en la investigación y aplicación del sistema pipeline HDRP de Unity3D, y su enfoque en la deformación de mallas 3D para la simulación de fluidos, técnica que permite representar cuerpos líquidos y su interacción con el entorno sin la necesidad de partículas individuales.
Finalmente, se observa el resultado de la combinación de cada enfoque técnico, tales como la integración de shaders y VFX con simulaciones de partículas, shaders con mallas HDRP, y la interacción entre HDRP y partículas realistas, con el objetivo de evaluar el potencial y las limitaciones de cada fusión en términos de rendimiento, calidad visual e interactividad.

Shaders y sistema de VFX

En el desarrollo de escenas tridimensionales, los shaders y los efectos visuales ocupan un rol central para alcanzar niveles de realismo que serían imposibles de lograr únicamente mediante geometría o texturas tradicionales. Un shader, entendido como un sistema, que se ejecuta directamente sobre la GPU y permite manipular aspectos como iluminación, color, transparencia, desplazamiento y comportamiento de los materiales, entre otros. Este procedimiento hace la tarjeta gráfica clave para mantener el rendimiento del motor, ya que la GPU está optimizada para ejecutar miles de tareas al mismo tiempo, evitando así cargar la CPU. Y hay que tener en cuenta que los efectos visuales, por su parte, pueden dividirse en dos grandes grupos según su origen y la forma en que interactúan con el motor, los VFX internos y los VFX externos.

VFX Internos

Los internos son aquellos creados directamente dentro de Unity, ya sea mediante herramientas como el Visual Effect Graph o el Particle System. Estas herramientas permiten construir sistemas complejos de partículas, simulaciones de humo, fuego, magia, distorsiones o recursos volumétricos, con la ventaja de estar renderizados a tiempo real. El Visual Effect Graph, en particular, trabaja de forma estrecha con el sistema de shaders de Unity y permite aprovechar la GPU para simular decenas de miles de partículas sin saturar la CPU, lo que lo convierte en una opción ideal cuando se necesita desarrollo rápido, optimización dentro del propio proyecto y capacidad de ajustar cada comportamiento sin salir del propio motor.

VFX Externos

En contraste, los VFX externos se desarrollan en programas especializados como Blender o Houdini, que ofrecen herramientas más avanzadas para simulaciones físicas complejas, fluidos realistas, humo animado o efectos cinematográficos. Blender, por ejemplo, permite generar efectos de humo o fuego extremadamente detallados sin necesidad de calcular toda la física en tiempo real. Este enfoque es especialmente recomendable cuando el efecto final no necesita ser interactivo o cuando el hardware objetivo no podría sostener la simulación en vivo.

Recomendacion según el objetivo

La elección entre un VFX interno o uno externo depende principalmente del grado de control, realismo y desempeño requerido en el proyecto. Los efectos internos son preferibles cuando quiero que sea dinámico en la jugabilidad, cuando buscamos reacción rápida o cuando se busca mantener una estructura integrada en el motor.

Los externos, en cambio, son la opción adecuada cuando se necesita un resultado visual altamente realista, cinematográfico o cuando las simulaciones superan lo que sería razonable renderizar a tiempo real dentro del juego.

Simulación mediante mallas 3D

El sistema de mallas deformables utilizado por HDRP permite representar superficies de agua con un nivel de realismo que sería muy costoso de lograr mediante partículas o simulaciones físicas. Al estar integrado directamente en el motor, evita la necesidad de recurrir a sistemas VFX externos.

Este tipo de simulaciones parte a partir de un plano o una malla y deformarla mediante desplazamientos calculados en tiempo real para generar la apariencia de olas, ondulaciones, corrientes y variaciones en la superficie. Esta aproximación es eficiente porque la malla en sí no cambia de complejidad, es la manipulación de una malla tridimensional mediante shaders especializados que se ejecutan en la GPU. solo se alteran sus vértices, lo cual permite mantener un rendimiento estable incluso en las superficies que simulen una gran extensión.

En Unity HDRP, aunque esté enfocado en el sistema de agua, esta técnica no se limita únicamente al agua. El uso de mallas deformables puede aplicarse a cualquier tipo de fluido cuya apariencia se base principalmente en deformaciones superficiales y no en un comportamiento volumétrico completo.
Por ejemplo, superficies de lava, geles o lodo pueden simularse de manera convincente mediante variaciones en los parámetros de ruido, desplazamiento y color, combinados con shaders que controlen la opacidad, la emisión o la refracción según el material.
También, efectos gaseosos densos, como nubes o niebla, pueden representarse mediante mallas, también si el objetivo visual es estilizado y no requiere simulaciones físicas.

Recomendacion según el objetivo

Esta técnica es especialmente recomendable en juegos donde se necesiten superficies amplias, como océanos, también puede utilizarse en otros fluidos como lava o sustancias semi-transparentes, cuando se busca un resultado visual estilizado.
Sin embargo, no debe emplearse para fluidos con volumen dinámico, como humo realista o líquidos que deban colisionar. La malla representa únicamente la superficie externa del fluido, por lo que pierde eficacia cuando el comportamiento visual requiere interacción del volumen interno.

Simulacion de particulas

La simulación de fluidos basada en partículas es uno de los métodos más representativos cuando se busca reproducir comportamientos físicos reales dentro de Unity. En este enfoque, el fluido se compone de muchas partículas individuales que interactúan entre sí a través de fuerzas de presión, viscosidad, cohesión y gravedad, donde cada partícula transporta propiedades físicas y contribuye de manera directa a la forma final del fluido. Esta aproximación permite reproducir fenómenos naturales como salpicaduras, fragmentación, turbulencia o la mezcla entre distintos líquidos, ofreciendo un nivel de dinamismo que otros sistemas no alcanzan.
En escencia, este tipo de simulación resulta mucho más cercano a un fluido real porque no se limita a deformar una superficie externa, sino que modela su movimiento desde dentro. Cada partícula posee propiedades como posición, velocidad, densidad y presión, que se actualizan en cada paso de simulación según la influencia de sus vecinas.

A pesar de que Unity no incluye un sistema oficial de fluidos realistas integrado en el motor, sí cuenta con su propio Particle System interno, pensado principalmente para efectos visuales simples (humo, chispas, fuego, polvo) y no para simulación física compleja. Además, existen múltiples implementaciones desarrolladas por terceros que buscan llevar la simulación de fluidos al entorno de Unity, utilizando diferentes enfoques y optimizaciones para CPU o GPU. Dentro de este conjunto de soluciones, una de las más relevantes —y la que abordaré específicamente en este trabajo— es la basada en la extensión del método SPH (Smoothed Particle Hydrodynamics) desarrollada en el documento “Particle-Based Fluid Simulation for Interactive Applications” de Müller et al. (2003), que permite construir un sistema de fluidos tridimensional físicamente coherente y aplicable a proyectos en tiempo real.

¿Qué es SPH?

El método SPH (Smoothed Particle Hydrodynamics) es un modelo de simulación donde un fluido se representa únicamente a través de partículas. A diferencia de las simulaciones basadas en mallas o superficies, SPH no necesita una estructura fija para describir la forma del fluido: la forma y el movimiento emergen directamente de las interacciones entre las partículas. Cada una de ellas posee información física como posición, velocidad, densidad o presión y se actualiza constantemente según la influencia de sus partículas vecinas.

La clave de este método está en el uso de los llamados kernels de suavizado, que funcionan como una regla que define qué partículas están “cerca” unas de otras y cómo contribuye cada una al comportamiento del fluido. En lugar de calcular fuerzas entre todas las partículas del sistema, lo cual sería imposible de sostener en tiempo real, los kernels establecen un radio de influencia: solo las partículas dentro de ese rango afectan a una partícula determinada. Es decir, una especia de burbuja que le permite al sistema ignorar todo lo que está lejos y concentrarse únicamente en el “pequeño grupo” que realmente importa. Gracias a esto, SPH puede simular líquidos complejos sin que la computadora explote tratando de calcular interacciones imposibles, logrando que el movimiento resultante sea suave y coherente, evitando saltos bruscos y permitiendo simular ondas, remolinos y deformaciones complejas sin depender de una malla rígida.

Müller y su implementación en Unity

El trabajo de Müller, Charypar y Gross (2003) marcó un antes y un después en el uso del método SPH dentro de entornos interactivos. Antes de su propuesta, SPH se utilizaba principalmente en contextos científicos y simulaciones de alta precisión, pero no estaba pensado para funcionar en tiempo real ni para integrarse en motores como Unity. Lo que Müller hizo diferente fue simplificar y optimizar el modelo matemático, proponiendo variantes de los cálculos de presión, densidad y fuerzas internas que reducían enormemente el costo computacional sin sacrificar estabilidad visual.

Su aporte permitió que SPH dejara de ser un método reservado para supercomputadoras o simulaciones offline, y pasara a ser una técnica viable para videojuegos y aplicaciones interactivas. Müller también presentó soluciones prácticas para reconstruir la superficie del fluido a partir de partículas, aprovechando los shaders y sistema de mallas, que permitían generar superficies suaves, continuas y compatibles con los sistemas de render.

Gracias a estas optimizaciones y a la forma modular en la que estructuró el método, SPH pasó a ser un sistema que se integra naturalmente con compute shaders en GPU, lo que facilita implementarlo en Unity y otros motores en tiempo real. En otras palabras, Müller no solo refinó el método desde lo matemático, sino que también lo adaptó a las necesidades reales del desarrollo interactivo, convirtiéndolo en una herramienta accesible, eficiente y compatible con las demandas de la simulación moderna.

Recomendacion segun el objetivo

Este enfoque hace que SPH se utilice principalmente cuando se busca un comportamiento interno realista del fluido. Como las fuerzas se calculan desde las propias partículas, el método es capaz de reproducir salpicaduras, turbulencias, fragmentación, mezcla de materiales y deformaciones libres, aspectos que serían difíciles de lograr con un sistema basado solo en desplazamiento de superficies o efectos visuales. En Unity, el método suele implementarse mediante compute shaders en la GPU, ya que estos cálculos requieren procesar un gran número de partículas de manera paralela. La GPU permite que SPH funcione en tiempo real, lo que hace posible utilizarlo tanto en simulaciones experimentales como en proyectos dinamicos que necesiten un líquido convincente, este metodo es funcional donde el movimiento interno del fluido define el resultado visual final. Por eso se aplica en simulaciones de agua en movimiento, chorros que se fragmentan, líquidos dentro de recipientes, mezclas entre fluidos y efectos donde el impacto o la colisión modifican la forma del fluido. Su flexibilidad también permite simular líquidos más densos, como lava o barro, ajustando propiedades como la viscosidad; incluso puede adaptarse a gases o humo si se combinan sus partículas con shaders volumétricos.

Comparación General de Métodos

A lo largo de este trabajo quedó claro que Unity no posee un único sistema oficial para simular fluidos, sino que ofrece herramientas que pueden combinarse para lograr soluciones diversas según el objetivo del proyecto. Esta flexibilidad es una ventaja, pero también obliga a entender qué ofrece cada enfoque y cuáles son sus límites para elegir correctamente.

Los shaders y VFX permiten construir fluidos estilizados o efectos visuales rápidos, con un costo muy bajo en rendimiento y una enorme libertad estética. Son ideales cuando la prioridad es la apariencia y no la física interna. Su desventaja es que dependen exclusivamente de trucos visuales, no existe volumen real, no hay interacción física entre partículas y el comportamiento del fluido no responde de forma dinámica al entorno.

El enfoque mediante mallas 3D deformables (HDRP Water System) se sitúa en un punto intermedio. La malla representa únicamente la superficie del fluido, pero ofrece interacción visual avanzada con la iluminación, refracción, profundidad óptica, entre otros. Es una solución ideal para océanos, lagos y ríos, donde el movimiento interno no es tan importante como el comportamiento superficial. Si bien puede adaptarse parcialmente para simular lava o líquidos densos, su mayor limitación es que no puede responder a colisiones internas ni al comportamiento como un fluido volumétrico real.

La simulación por partículas físicas, es la más completa en términos de comportamiento realista, porque modela el fluido desde adentro hacia afuera. El movimiento emerge de la interacción entre partículas, permitiendo fragmentación, mezcla, turbulencias y deformaciones libres. Su mayor ventaja es la precisión; su principal desventaja, el costo computacional. La implementación correcta exige aprovechar la GPU mediante compute

shaders, ya que la CPU no puede manejar miles de partículas en tiempo real sin comprometer el rendimiento.

Respuesta a las Preguntas de Investigación

¿Qué tan complejos y eficientes pueden llegar a ser los fluidos simulados en Unity?
Pueden ser muy complejos, especialmente si se utilizan técnicas basadas en SPH. Sin embargo, la eficiencia depende directamente de la GPU y del número de partículas. Unity permite fluidos realistas en tiempo real siempre que se optimice el cómputo. La complejidad visual puede alcanzarse mediante shaders y reconstrucción de superficie, incluso cuando la simulación es relativamente liviana.

¿Cómo se pueden optimizar para su uso en tiempo real sin perder calidad visual?
Las partículas pueden simularse con menos precisión, mientras que un shader de superficie puede suavizar y mejorar el fluido. Para cuerpos de agua grande, HDRP Water System es más eficiente que SPH porque la superficie se calcula directamente en GPU sin simular volumen interno. Pero el uso de VFX, sera en su mayoría, el metodo principalmente usado en casos que se busque eficiencia y calidad visual.

¿Cómo se relaciona la simulación de fluidos con la iluminación volumétrica?
Los fluidos, especialmente los simulados por partículas, producen densidades internas que la iluminación volumétrica puede aprovechar. Humo, vapor o niebla requieren volumen para dispersar la luz. En agua, HDRP utiliza profundidad óptica y absorción lumínica. En partículas SPH, la densidad puede influir en la refracción, el color, la sombra interna del fluido y la interacción con luces volumétricas. Y en VFX, la iluminacion tendra una iteracion segun los parametros que se decidan utilizar.

¿Qué técnicas o herramientas permiten integrar fluidos y luz de forma coherente?
Los shaders personalizados, el Visual Effect Graph, HDRP Water System y los métodos de reconstrucción de superficie del SPH permiten integrar partículas y luz. Los fluidos volumétricos, como humo, se conectan bien con la iluminación volumétrica nativa del motor, mientras que los fluidos superficiales, agua o lava, necesitan shaders que simulen refracción o profundidad. La elección depende del tipo de fluido y del renderizado .

Simulación en CPU vs GPU

En fluidos simulados físicamente (como SPH) la diferencia es crítica.
Simular interacciones partícula–partícula en CPU provoca un cuello de botella inmediato.
La GPU, en cambio, puede procesar miles de partículas en paralelo. Como referencia, según las prácticas, la GPU puede manejar hasta 50.000 partículas en tiempo real, mientras que la CPU apenas alcanza unas pocas centenas con buena estabilidad.

En métodos por mallas, la GPU también es esencial porque la deformación de vértices se realiza mediante shaders. En VFX, prácticamente toda la carga recae en GPU.

Aunque la GPU suele ser la opción preferida para la mayoría de las simulaciones de fluidos, la CPU sigue siendo útil en casos específicos. La ventaja principal de la GPU reside en su arquitectura que está diseñada para ejecutar miles de operaciones simultáneas, algo ideal para cálculos donde cada partícula debe evaluarse junto con sus vecinas. Por eso, en métodos como SPH o simulaciones basadas en grandes volúmenes de partículas, la GPU no solo es más rápida: es prácticamente la única forma viable de mantener la simulación en tiempo real.La CPU, por otro lado, resulta más conveniente cuando la simulación requiere lógica compleja como interacciones con el gameplay y en escenas pequeñas, donde la cantidad de partículas es muy baja y priorizar la GPU no ofrece una mejora real.

Conclusión Final

La clave, para crear el sistema de fluidos adecuado, no está en elegir un único método, sino en combinar el adecuado según el objetivo del proyecto.
Si se busca un fluido realista y dinámico que reaccione al entorno, SPH es la opción más poderosa.
Si la prioridad es simular grandes superficies con iluminación compleja, HDRP ofrece soluciones de alta calidad con un costo menor.
Para efectos estilizados o livianos, los VFX internos son la opción ideal, mientras que los externos permiten obtener resultados cinematográficos sin comprometer el rendimiento en tiempo real.

La combinación de estos tres métodos permite crear sistemas de fluidos híbridos capaces de aprovechar lo mejor de cada enfoque. Por ejemplo, un SPH puede simular el volumen interno del agua mientras un shader reconstruye su superficie de forma eficiente; una malla HDRP puede representar un océano lejano mientras un sistema de partículas gestiona espuma o salpicaduras en zonas cercanas; o un VFX volumétrico puede añadirse sobre un fluido físico para mejorar su interacción con la iluminación.
El resultado es un sistema más completo, expandible y visualmente coherente, donde cada componente cumple un rol específico sin sobrecargar el rendimiento o limitarse a un metodo.

Avances para re-entrega

Desde hace unos días estuve revisando por qué no estaba funcionando la simulación de partículas de agua en mi proyecto. Yo estaba usando como referencia un proyecto hecho en Unity con Built-in Render Pipeline o URP, y al intentar llevarlo a mi proyecto en HDRP la simulación directamente no aparecía en escena.

Después de analizar el problema, entendí que no estaba fallando el cálculo del sistema, sino la parte visual. Los compute shaders se ejecutan de la misma forma en HDRP, pero el

problema está en que los shaders y los métodos de render utilizados en el proyecto original no son compatibles con el pipeline HDRP. Esto hace que los materiales no se instancien correctamente y que los datos calculados en la GPU no lleguen a mostrarse en pantalla.

En resumen, el conflicto no está en la simulación en sí, sino en la forma de renderizar el resultado dentro de HDRP, que trabaja con un sistema de shaders y un pipeline distinto.

Soluciones viables

El método que se utilizare para solucionar el problema encontrado sera crear un nuevo proyecto de Unity utilizando el Universal Render Pipeline (URP) en lugar del High Definition Render Pipeline (HDRP), ya que esto permitirá utilizar correctamente todo lo que proporciona el proyecto de referencia de partículas SPH (Smoothed Particle Hydrodynamics) para simular fluidos; al mismo tiempo, este enfoque facilitará centrarse en el sistema SPH para entender, aprender a utilizar y conocer a mayor profundidad su funcionamiento.

Explicaciones de lo investigado.

En esta parte del texto quiero dejar claro que voy a hablar de las investigaciones que hice para poder entender por qué lo programado tienen que ser así para que funcione y qué pasaría si no están puestas como están ahora.

Lo que más me sorprendió al principio fue el manejo de bytes al serializar los datos en la GPU, ya que es crucial entender la cantidad exacta de información que estás creando con variables como float o vectores, esta información se declara previamente en la memoria y, al transmitir los datos de la GPU a la CPU mediante un buffer, este debe tener exactamente la misma cantidad de bytes por partícula para que la información se transfiera correctamente. Si se permite que C# o Unity modifiquen esta asignación de memoria por optimización, existe un riesgo significativo de que se produzcan errores, como posiciones incorrectas o la transferencia de información corrupta a mitad del proceso.

Para iniciar las partículas, el proceso usa como referencia una esfera low poly. Esta esfera se emplea para el renderizado instanciado, en lugar de hacer miles de llamadas individuales para renderizar cada partícula, se realiza una sola llamada que dibuja todas las esferas, basándose en la posición y vectores de cada una. Como cada partícula ocupa 44 bytes, se crea un espacio de memoria (buffer) que contiene la información de todas ellas, finalmente este buffer se conecta con los kernels del Compute Shader, que son los encargados de leer y escribir los datos de las partículas durante la simulación.

Posteriormente, se configura la comunicación entre C# y el código que se ejecuta en la GPU. Esto se logra obteniendo primero los kernels (similares a funciones) y enviando luego los parámetros copiados desde C# hacia la GPU. Estos parámetros actúan como variables globales que el shader puede leer, incluyendo datos como el número de partículas, el tamaño de la simulación, la masa o la viscosidad. También se calculan las potencias del radio las cuales no llegue a investigar bien a fondo, solo con conocimiento superficial de que ahorra tiempo de procesamiento. Finalmente, se conectan los buffers de partículas, que

previamente aclare que contiene los datos de cada particula, para que los kernels puedan leer y modificar los datos de cada una.

Tambien se debe configurar el contacto de la física de las partículas con el objeto central (la esfera), no estamos realmente simulando una colisión con el modelo 3D, sino con las matemáticas que definen el espacio que ocupa ese objeto. Para lograrlo, la simulación solo necesita la posición (obtenida del `Transform` de Unity) y el radio de la esfera. Por eso, al experimentar con la escena 1, donde agrego una caja con colision sin usar directamente la variable `Transform`, dejandosela a la esfera, sin que ocurra nada y luego en la escena 2, donde la opción de intercambio muestra una forma esférica dentro del cubo, se confirma que la interacción se basa puramente en un cálculo matemático de volumen esférico, no en la geometría visible del modelo. Como dato extra en la escena 5, no solo esta desactivada la funcion Sphere Collider, tambien Mesh Collider, confirmando que solo se basa en calculos de posicion y tamaño, porque ademas, las particulas realmente, no tienen colision como con la funcion de unity de Collider, por eso atraviesan el cubo.Con la simulacion de que hay piso, pero solo son una ilusion de caja, a base de Vectores, que el Compute Shader, lo utiliza para saber cuando va a rebotar con las “paredes”

Finalmente, comentare lo que entendí de los Compute Shaders utilizando tres kernels (que son como funciones dentro de la GPU), uno de ellos procesa 100 partículas a la vez en paralelo. Este kernel se encarga de calcular la densidad y la presión de cada partícula, para ello, revisa cuántas partículas hay en su cercania para determinar qué tan comprimida está, lo que a su vez define la fuerza con la que empuja hacia afuera. Ademas, como en el caso de estas simulaciones, si se instancian 1000 particulas, se crean 10 grupos de 100 particulas que procesan en paralelo.

El segundo kernel se encarga de calcular las fuerzas internas clave del fluido, la gravedad que simula el peso real y empuja hacia abajo, la presión que hace que las partículas se separen al estar comprimidas, manteniendo el volumen, la viscosidad que simula el arrastre entre ellas y la colisión con esferas actúa como una cuarta fuerza o mecanismo de empuje que interviene asegurar que se no invadan el espacio de los límites externos de cada esfera

El tercer kernel, es el encargado de mover las partículas. Utiliza la fuerza total calculada (gravedad, presión, viscosidad) para cambiar la velocidad de la partícula, lo que a su vez modifica su posición. Además, en este paso se aplica la restricción de límites definida por la caja, creada a partir de vectores, esta restricción provoca un rebote imperfecto al aplicar una variable que hace que la partícula pierda velocidad con cada colisión, simulando de manera más realista una partícula que pierde energía.

Todo este proceso nos permite lograr una simulación de mil esferas muy realista y altamente optimizada gracias al uso del Compute Shader. Este código simula la física de la gravedad, la viscosidad y otras fuerzas de empuje que modifican la velocidad y la posición de las partículas. Además, la gran ventaja se evidencia al compararlo con el sistema nativo de Unity, la Escena 3 muestro que simular una pequeña fracción de estas partículas ,con niveles del 1 a 5, con componentes como Rigidbody y Colliders es mucho más costoso en términos de procesamiento, ya que con el método de Unity cada esfera requiere que el motor calcule la colisión individualmente con el piso, las paredes y la esfera central,

mientras que el shader lo resuelve de forma masiva y matemática. Ademas del renderizado de cada una de estas esferas de forma individual.

Finalmente, en las Escenas 4 y 5 realicé varias pruebas ajustando los valores internos de la simulación. Modifiqué la viscosidad, el tiempo de reacción (o timestep), la pérdida de velocidad al rebotar en las paredes (damping), la masa de las esferas (que afecta la gravedad), la separación inicial de las esferas en reposo y, crucialmente, el factor que define qué tan rígido o suave se comporta el fluido ,como agua o gas.

Bibliografía

Daily.dev. (n.d.). *Unity fluid simulation tutorial: CPU and GPU methods*. https://daily.dev/blog/unity-fluid-simulation-tutorial-cpu-and-gpu-methods

MinionsArt. (2021). *Fake fluid shaders in Unity* [Video]. YouTube. https://www.youtube.com/watch?v=XP7jDuyB2FA

Müller, M., Charypar, D., & Gross, M. (2003). *Particle-Based Fluid Simulation for Interactive Applications*. https://matthias-research.github.io/pages/publications/sca03.pdf

Unity Technologies. (2019). *Fluid Simulation in Unity* [Video]. YouTube. https://www.youtube.com/watch?v=zbBwKMRyavE

Unity Technologies. (n.d.). *High Definition Render Pipeline (HDRP) Water System – Usage*. https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@14.0/manual/WaterSystem-use.html

Unity Technologies. (n.d.). *Shader Graph manual*. https://docs.unity3d.com/Packages/com.unity.shadergraph@latest

Unity Technologies. (n.d.). *Visual Effect Graph manual*. https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@latest

Unity Technologies. (n.d.). *Unity Particle System manual*. https://docs.unity3d.com/Manual/PartSysMainModule.html