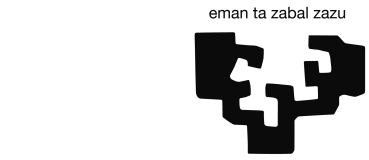


Grado en Ingeniería Informática

Especialidad en Computación

Proyecto de Fin de Grado

# *Simulación de Fluidos con GPGPU*

Autor: Andoni Rivera Pinto

Directora: Carmen Hernández Gómez


Septiembre 2018

# Agradecimientos

Tras un largo periodo de estudio y desarrollo del proyecto tanto con sus partes positivas como con sus dificultades técnicas, me gustaría agradecer a mi directora del proyecto Carmen Hernández toda la ayuda y el apoyo ofrecido durante estos meses además de por haberme brindado las herramientas necesarias para completar el proyecto, por las asignaturas en las que ha sido mi profesora.

Por otra parte me gustaría agradecer a mi familia por haberme apoyado durante toda la carrera y haber estado ahí en los momentos de mayor dificultad.

Por último, querría agradecer a mis amigos gracias a los cuales he sido capaz de desconectar en momentos de tensión y charlar sobre temas interesantes relacionados con la informática.

# Resumen

Los fluidos juegan un papel interesante en el mundo de la simulación. Los efectos de agua en películas, la simulación de cuerpos líquidos en videojuegos, las simulaciones de escenarios de la vida real como son las catástrofes naturales (inundaciones, tsunamis, comportamiento de ríos y lagos,...). Todos ellos son posibles gracias al concepto conocido como GPGPU y el procesado en paralelo mediante la unidad de procesamiento gráfico. En este proyecto, se realiza el estudio de la simulación de líquidos mediante GPGPU en el cual se estudian las ecuaciones que gobiernan los fluidos en el mundo real también conocidas como ecuaciones de Navier-Stokes, el estudio de distintos métodos de integración mediante los cuales se podrá implementar una simulación basada en las ecuaciones de fluidos, la implementación de la simulación basada en lo previamente estudiado mediante GLSL, el análisis de las herramientas que se van a utilizar para ello y el análisis de sus resultados.

# Índice general

# Índice de figuras

# 1. CAPÍTULO

## Introducción

Durante las últimas décadas, el mundo de los gráficos por computador ha buscado insistentemente el modo de simular aspectos propios de la vida cotidiana. Dentro de estas líneas de investigación, uno de los campos que ha experimentado una gran proyección ha sido la simulación de fenómenos naturales. Su estudio, mediante las técnicas de simulación dinámica, es posible cada vez con más exactitud gracias, en parte, a los avances en la capacidad de computación. Es este hecho el que permite, además, un análisis cada vez más detallado y preciso de sus causas y consecuencias, con el fin de evitar los desastres humanos y económicos que se pudieran generar (terremotos, explosiones, cambio climático, volcanes, tormentas, grandes mareas, etc.).

De entre todos estos fenómenos, uno de los más estudiados es el modelado y animación de fluidos. Esto se debe, sin duda, a la gran complejidad que conlleva la simulación de un fluido en términos físicos, ya que no sólo hay que tener en cuenta las causas o el modo en el que se produce el movimiento del líquido, sino que además hay que realizar una simulación tan realista como se pueda para que su análisis pueda ayudar a la comprensión de su compleja dinámica.

Gotas de lluvia, olas de mar, ríos de lava, cascadas que caen de una colina,... Son muchas las situaciones en las que nos podemos encontrar un fluido, y no es de extrañar que se quiera incluir animaciones realistas e impresionantes en películas o largometrajes, videojuegos o spots comerciales.

**Figura 1.1:** Aspersores situados en una zona alta para simular la lluvia.

No fue hasta finales de los 90 cuando comenzaron a utilizarse efectos por ordenador. Antes, la única forma de obtener efectos líquidos creíbles era recrear el efecto líquido en el escenario.

Sin embargo, este recurso cuenta con sus limitaciones. Inicialmente porque a menudo se tiende a exagerar la realidad o crear efectos físicamente imposibles. Como segundo motivo, está el de la seguridad. La recreación del efecto líquido puede ser demasiado peligroso en escenas como puede ser la simulación de un maremoto. Otro motivo es el gran coste que supone la simulación de efectos a gran escala. Se han intentado reducir costos mediante la creación de escenarios a pequeña escala. Desgraciadamente el agua no se comporta a escala y solo se pueden crear efectos de agua creíbles si el factor de escala no es inferior a aproximadamente una cuarta parte.

Este tipo de simulaciones no es algo trivial. El comportamiento de los líquidos forman ondas, crean espuma, dos volúmenes de un líquido pueden fusionarse, un líquido puede dividirse en gotas, etc.

Las formas de representar efectos líquidos digitales puede clasificarse en dos categorías: emulación y técnicas de simulación. Los métodos de emulación intentan reproducir el comportamiento y la apariencia del flujo de líquido sin modelar la física subyacente, lo cual dificulta la labor de capturar la dinámica de los líquidos y también se restringe a un efecto particular.

Una forma más natural de modelar líquidos es usar las ecuaciones de Navier-Stokes, que son las ecuaciones físicas que describen su movimiento.

Estas técnicas de simulación pueden crear efectos muy realistas y proporcionar un modelo

general. Sin embargo, es más difícil controlar el flujo de líquido porque está dictado por la física de fluidos.

Podemos dividir en dos categorías las técnicas de simulación de líquidos que son los métodos eulerianos y lagrangianos. Como punto en común está que se resuelven mediante las ecuaciones de Navier-Stokes. Los métodos eulerianos (basados en la cuadrícula) calculan la velocidad del líquido en puntos fijos del espacio y los métodos lagrangianos calculan la velocidad del líquido en los puntos que se mueven junto con el líquido. Los métodos eulerianos resuelven problemas de pequeña escala y los métodos lagrangianos no tienen tales problemas de pérdida de masa y pueden manejar problemas de escala mediana pero con el inconveniente de representar superficies lisas de líquido.

A continuación vamos a comentar sobre la simulación de líquidos y distintas formas de representación populares que se utilizan hoy en día.

## 1.1. Objetivos

Este proyecto tiene como objetivo principal la creación de un sistema que simule el comportamiento del agua.

Para ello, es necesario el estudio de las distintas formas de simulación de los fluidos a lo largo del tiempo y el estudio de los aspectos matemáticos y físicos que juegan un papel crucial en la simulación y el renderizado de los fluidos; en particular, una superficie de agua.

También se plantea como un objetivo específico de este proyecto, el estudio de la librería de WebGL conocida como “Three.js”. Una vez realizado un estudio de las diferentes técnicas de modelado y animación de fluidos existentes en la literatura, diseñamos una aplicación basada en dicha técnica; en nuestro caso, la utilización de mapas de alturas “height fields”. Después de realizar la implementación en Three.js, diseñamos los casos de prueba y analizamos los resultados obtenidos tras diferentes ejecuciones de la aplicación.

## 1.2. Motivación

El motivo de haber escogido este proyecto frente a otros relacionados con la rama de la computación es la afición que tengo al mundo de los videojuegos y de la animación. En

**Figura 1.2:** Simulación de agua de mar en la película Vaiana de Disney.

la actualidad, existen muchas aplicaciones de animación en tiempo real que pretenden simular el mundo en el que vivimos a través de las leyes físicas. Por ejemplo, películas de animación como "Vaiana" realizada en 2016, donde una parte importante de la película se desarrolla en el mar y, gracias a la tecnología actual y a los algoritmos de simulación es posible darle un toque realista a una historia ficticia.

Por otra parte, la simulación de fluidos es parte importante de la visualización científica así como la simulación en escenarios reales tales como el análisis hidrológico de ríos y la gestión de emergencias. Por ejemplo, mediante este tipo de aplicaciones puede estimarse qué ocurriría en el caso de que una presa rompiese e inundase sus alrededores.

En resumen, tal vez a muchas personas les parezca solamente atractivo el producto final. Sin embargo, la curiosidad de lo que hay detrás de este tipo de programas o simuladores es lo suficiente grande como para que me interese profundizar en su diseño e implementación.

## 1.3. Contenidos de la memoria

En esta Memoria del proyecto de Fin de Grado, hemos incluido un capítulo que recoge el "Documento de los objetivos del proyecto" y que consta de varios apartados en los que se explican las competencias a realizar en este proyecto. En el siguiente capítulo, titulado "Estado del arte", presentamos los trabajos más relevantes que hemos encontrado

dentro del área de la simulación en tiempo real de fluidos. Posteriormente, hemos incluido el capítulo de “Diseño” en el cual se describe el diseño de la aplicación que hemos realizado y que resuelve el objetivo principal del proyecto. Una vez descrito el diseño de la aplicación a realizar, presentamos un capítulo denominado “Implementación” que incluye una descripción detallada de la aplicación en el entorno de Three.js para la simulación y animación del agua utilizando capacidades de cómputo de una GPU a partir de la computación conocida como GPGPU. También hemos realizado distintas pruebas y hemos analizado los resultados de dichos casos de prueba que incluimos en el capítulo de “Resultados”. Finalmente presentamos las conclusiones y líneas futuras así como la bibliografía utilizada en el desarrollo de este proyecto.

# 2. CAPÍTULO

## Documento de los objetivos del proyecto

En este capítulo se definen todos los elementos a tener en cuenta en el proyecto que son, los objetivos que se pretenden alcanzar en el proyecto, las herramientas que se van a utilizar para cumplir dichos objetivos, la planificación a seguir para cumplir con los objetivos de forma adecuada y el análisis de riesgos y la factibilidad del proyecto.

### 2.1. Objetivos del proyecto (alcance y exclusiones)

Los objetivos principales de este proyecto son los siguientes:

1. Estudio de los distintos modelos de representación existentes para la simulación y animación de fluidos.
2. Estudio de los aspectos matemáticos y físicos de este tipo de modelos.
3. Elección de uno o varios modelos que encajen con el proyecto que se desea desarrollar y estudiar sus comportamientos.
4. Estudio y utilización de la librería de WebGL conocida como “Three.js” para la implementación de la aplicación de simulación de fluidos.
5. Implementación y análisis de una aplicación que simule el comportamiento del agua.

Los objetivos a cumplir en este proyecto se pueden dividir en dos subgrupos. Los primeros tres puntos corresponden al estudio y documentación, y el cuarto y quinto punto corresponden a la parte de implementación y análisis de la aplicación.

## 2.2. Herramientas utilizadas

En este apartado se recogen todos los entornos que se han utilizado para el desarrollo completo del proyecto. Hemos considerado que estas herramientas son las más adecuadas para el desarrollo del proyecto y el aprendizaje de uso de las mismas no impide que se cumplan los tiempos estimados. Por otra parte, el desarrollo de una aplicación ejecutable en un navegador web posibilita su ejecución en distintos sistemas operativos.

### 2.2.1. Three.js

Three.js es una biblioteca liviana de WebGL escrita en JavaScript para crear y mostrar gráficos animados por ordenador en 3D en un navegador Web. De hecho, es una de las más importantes para la creación de las animaciones en WebGL. Además, esta librería es un archivo independiente de JavaScript y puede ser incluido dentro de una página web a través de un enlace a una copia local o remota.

En la página de referencia de la librería Three.js, https://threejs.org/, se puede encontrar documentación *online* y distintos ejemplos de uso.

### 2.2.2. GLSL

*OpenGL Shading Language* o GLSL es un lenguaje de alto nivel de sombreado con una sintaxis basada en el lenguaje de programación C. Fue creado para ofrecer a los desarrolladores un control más directo en el proceso de renderizado sin tener que utilizar lenguaje ensamblador ARB o lenguajes específicos del hardware. Al igual que el lenguaje de programación C, GLSL admite bucles, condicionales, etc.

Un *shader* es un programa o algoritmo que se encarga del procesamiento de vértices (*vertex shader*) y de píxeles o fragmentos (*fragment shader*) y da flexibilidad al programador ya que permiten realizar transformaciones y crear efectos especiales tales como iluminaciones más precisas, fuego o niebla. Los *shaders* GLSL no son aplicaciones independien-

tes y requieren de una aplicación que utilice la API de OpenGL, que está disponible en distintas plataformas.

Como hemos dicho anteriormente, existen dos tipos principales de *shaders*:

1. *Vertex shader*: Es una función que se ejecuta para cada vértice de la geometría y que realiza operaciones matemáticas o de modificación de los atributos del vértice tales como cambios en la posición, en la normal, cálculo del color o incidencia de la luz en dicho vértice.

2. *Fragment shader*: También conocido como "*pixel shader*" es una función a nivel de píxel que permite realizar efectos gráficos con mayor precisión. También permite utilizar técnicas que no son posibles de realizar a nivel de vértice o que resultan menos precisas o efectivas si se realizan por medio de los *vertex shaders* como por ejemplo la iluminación basada en la física, la generación de sombras, etc.

### 2.2.3. Lyx

Lyx es un procesador avanzado de documentos de código abierto que funciona en Linux/Unix, Windows y Mac OS X. Se llama "procesador de documentos" porque, a diferencia de los procesadores de texto habituales, LyX apuesta por un enfoque basado en la estructura del documento, y no en su apariencia lo cual permite concentrarse en la escritura, dejando para el software los detalles del diseño visual. LyX automatiza el formato de acuerdo con un conjunto de reglas predefinidas, brindando consistencia completa incluso en los más complejos documentos. LyX genera salida de alta calidad, profesional usando en segundo plano LaTeX, un motor de potencia industrial para tipografía y de código abierto.

**Figura 2.1:** Logotipo del programa Lyx.

### 2.2.4. Firefox

Mozilla Firefox es un navegador web libre y de código abierto desarrollado para Linux, Android, IOS, OS X y Windows coordinado por la Corporación Mozilla. Usa el motor Gecko para renderizar páginas web, el cual implementa actuales y futuros estándares web. A través de este navegador, será posible la ejecución del simulador que se va a realizar en este proyecto.

**Figura 2.2:** Logotipo del navegador Mozilla Firefox.

## 2.3. Planificación

La planificación del proyecto ha sido bastante simple en cuanto a bloques de actividades se refiere.

Inicialmente, las primeras horas fueron dedicadas a tiempo completo al estudio y lectura de documentación de simulación de fluidos en tiempo real. Posteriormente, siguiendo con la actividad anterior, se dedicó tiempo a la formación de la herramienta Three.js ya que nunca había trabajado con ella.

Las siguientes semanas fueron para el desarrollo de la memoria, en la cual decidimos qué modelo se iba a implementar y se detallaron algunos elementos más. Además, se continuó con la lectura de más documentos con el fin de completar adecuadamente esta fase.

Finalmente se realizó la implementación y pruebas de su funcionamiento para concluir el proyecto. Tras realizar la implementación, se ha terminado la memoria con la redacción del análisis de resultados y las conclusiones.

## 2.4. Análisis de riesgos

Los riesgos que existen en este proyecto son:

1. Vacaciones de verano: La pérdida del ritmo de trabajo académico puede influir en el tiempo dedicado semanalmente al proyecto, lo cual puede suponer un retraso de éste y hacer inviable una entrega para la fecha deseada.

2. Problemas en las herramientas de desarrollo: El hecho de no poder continuar desarrollando el proyecto debido a un fallo en las herramientas utilizadas.

## 2.5. Análisis de factibilidad

Este proyecto es factible debido a que los riesgos que se han comentado en el punto anterior son controlables. El riesgo de las vacaciones de verano requiere únicamente una implicación total al proyecto para finalizarlo lo antes posible y el establecer hitos parciales para cumplirlos. En cuanto al segundo riesgo, se cuenta con un PC de respaldo con el cual, en caso de ser necesario y en caso extremo, se puede recurrir al material universitario que se encuentra disponible para los alumnos.

# 3. CAPÍTULO

## Estado del arte

La simulación de fluidos tiene una larga historia en el mundo de los gráficos por computador y ha atraído a cientos de investigadores en las últimas décadas. El modelado de fenómenos naturales como el agua sigue siendo un problema abierto en este ámbito ya que la dinámica de los fluidos es una tarea muy compleja.

La literatura en la dinámica de fluidos computacional (*Computational Fluid Dynamics*, CFD) contiene cientos de artículos sobre la simulación de fluidos y existen un gran número de enfoques que van desde los métodos espectrales a los elementos finitos. Los factores que guían a los investigadores a la hora de seleccionar los métodos de simulación de fluidos son: la facilidad de implementación, coste computacional bajo, posibilidad de control de la animación, incorporación de obstáculos y representaciones de superficies libres como el océano.

Los modelos visuales de fluidos tienen muchas aplicaciones en la industria, incluidos los efectos especiales en el mundo del cine y los juegos interactivos. Generalmente este tipo de simulaciones se realiza de forma *off-line* ya que el costo computacional es excesivamente alto. Es por ello que un gran número de investigaciones se centran en el estudio de algoritmos más eficientes y eficaces para la simulación y visualización de gases y fluidos como el agua.

Hay dos enfoques básicos para resolver las ecuaciones de fluidos: el enfoque basado en la red (euleriano) y el basado en partículas (lagrangiano). Ambos se ha utilizado con éxito para crear efectos impresionantes tanto en películas como en anuncios comerciales.

**Figura 3.1:** A simple model of ocean waves. Alain Fournier & William T. Reeves. 1986

Producir simulaciones más rápidas y mejorar la calidad o mantenerla a niveles similares ha sido el enfoque principal de los investigadores en este campo [Kontaxis, 2013].

El comportamiento del volumen de un líquido puede ser descrito por un conjunto de ecuaciones desarrolladas por Claude-Louis Navier y George Gabriel Stokes en el siglo XIX. En los últimos cincuenta años se ha dado un enorme esfuerzo de investigación por parte de la comunidad de CFD en resolver estas ecuaciones para ser aplicadas en diversas aplicaciones de ingeniería.

Los trabajos previos sobre la animación y modelado de fluidos, [Fournier and Reeves, 1986], se concentraban en modelar solamente la superficie del cuerpo líquido como una función paramétrica en el tiempo para simular una ola de transporte (Figura 3.1). Peachey [Peachey, 1986] presentó en el mismo año un trabajo similar (Figura 3.2).

Kass y Miller [Kass and Miller, 1990] adoptaron una aproximación diferente a la simulación de fluidos ya que utilizaron una superficie dinámica formada por campos de altura los cuales interactuaban con un “objeto” sólido y estático que simulaba el terreno (Figura 3.3). Al contrario que otros métodos, utilizaban una formulación basada en ecuaciones diferenciales parciales para el movimiento del agua donde dichas ecuaciones controlaban la cantidad de fluido que pasaba entre las columnas de agua.

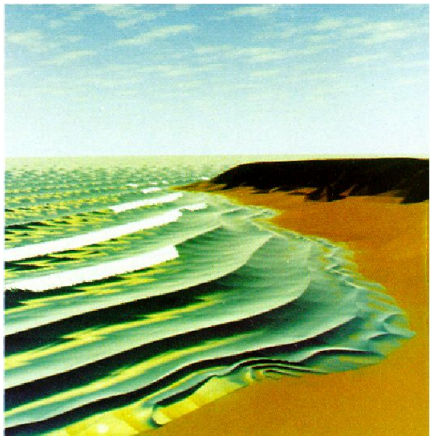

**Figura 3.2:** Modeling waves and surf. Darwyn R. Peachey. 1986

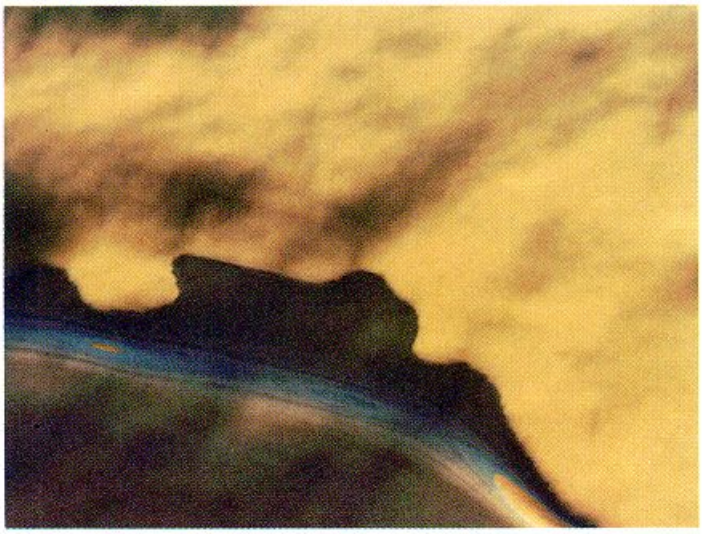

**Figura 3.3:** Rapid, stable fluid dynamics for computer graphics. Michael Kass & Gavin Miller. 1990

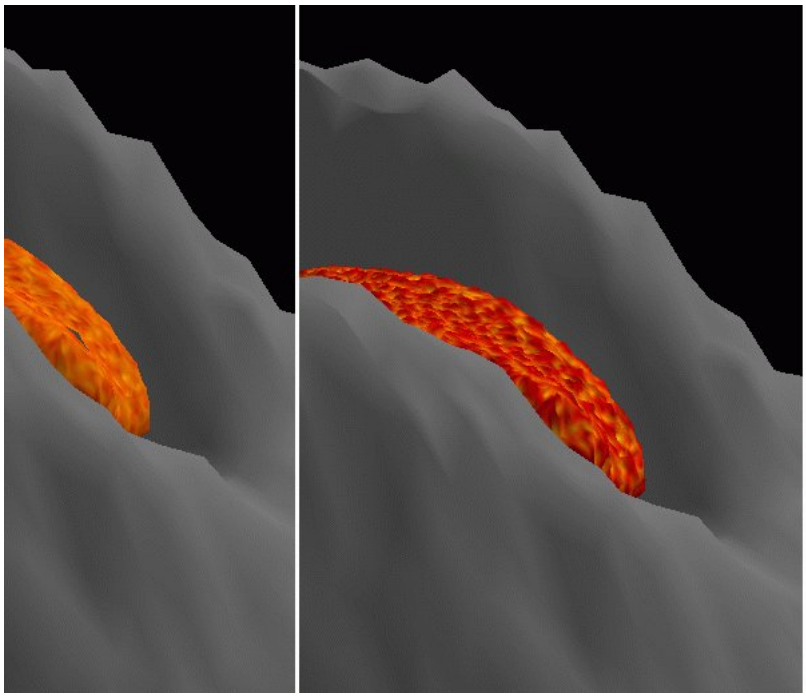

**Figura 3.4:** Animating Lava Flows. Stora et al. 1999

O'Brien y Hodgins [O'Brien and Hodgins, 1995] simularon salpicaduras de líquidos combinando un sistema híbrido de representación basado en partículas y en un campo de alturas. Por otra parte, Miller y Pearce [Miller and Pearce, 1989] utilizaron sistemas elásticos entre las partículas para conseguir un flujo dinámico en tres dimensiones. Terzopoulos, Platt y Fleischer [Terzopoulos et al., 1991] realizaron simulaciones de sólidos deformables utilizando una aproximación de dinámicas moleculares para simular las partículas en la fase líquida. Stora et al. [Stora et al., 1999] utilizó partículas y una aproximación al cálculo de fuerzas llamado *Smoothed Particle Hydrodynamics* (SPH) para simular un flujo de lava (Figura 3.4).

Los métodos basados en superficies o partículas son relativamente rápidos, especialmente en el caso de grandes volúmenes de líquido, pero no consiguen reproducir el rango de movimientos exhibidos por los líquidos. Especialmente, no aprovechan el realismo que ofrece una solución completa de las ecuaciones de Navier-Stokes y no pueden ser fácilmente adaptados para incluir la interacción de fluidos con objetos en movimiento.

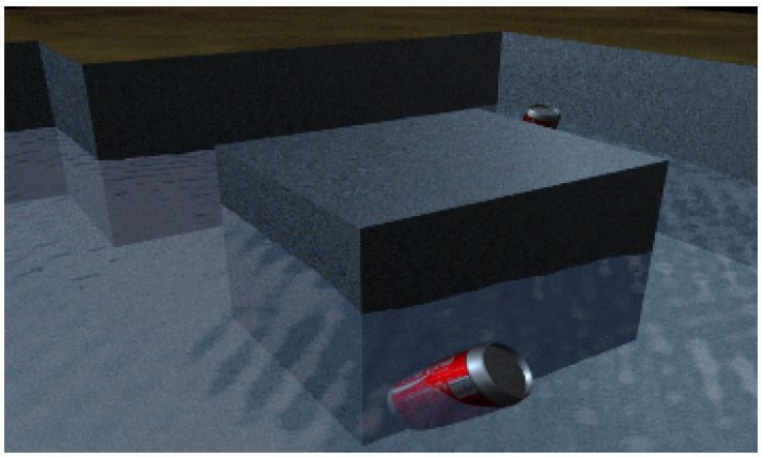

**Figura 3.5:** Realistic Animation of Liquids. Nick Foster & Dimitri Metaxas. 1996

El primer uso de esta aproximación en el mundo de los gráficos fue realizada por Chen y da Victoria Lobo [Chen and da Vitoria Lobo, 1995] ya que utilizaron una técnica de relajación para resolver las ecuaciones de Navier-Stokes en dos dimensiones y, mediante un campo de alturas basado en la presión, obtenían la tercera dimensión. Witting [Witting, 1999] utilizó un sistema en el cual las ecuaciones bidimensionales de Navier-Stokes eran incluidas en un entorno de animación. Dicho sistema permitía a los animadores crear y controlar efectos 2D como remolinos de agua y la expansión de humo. También utilizó un conjunto de ecuaciones que incluían difusión del calor y flotabilidad térmica, y un esquema Runge-Kutta de cuarto orden para resolver dichas ecuaciones.

Foster y Metaxas [Foster and Metaxas, 1996] popularizaron la resolución tridimensional de las ecuaciones de Navier-Stokes ya que modificaron el método original de Harlow y Welch [Harlow and Welch, 1965] para resolver las ecuaciones completas en tres dimensiones con objetos estáticos aleatorios y lo extendieron para incluir mecanismos de control simples. En una serie de publicaciones demostraron que la aproximación de Harlow y Welch podía utilizarse para animar agua, humo, y mejorarse para controlar el comportamiento de fluidos animados. La mayor ventaja de este sistema es que el líquido no está restringido a ser un mapa de alturas y que resuelven las ecuaciones de Navier-Stokes mediante diferencias finitas (Figura 3.5).

En cuanto a los métodos numéricos de computación podemos citar a los métodos lagrangianos tales como «*Smoothed particle hydrodynamics*», «*Moving least squares*» y

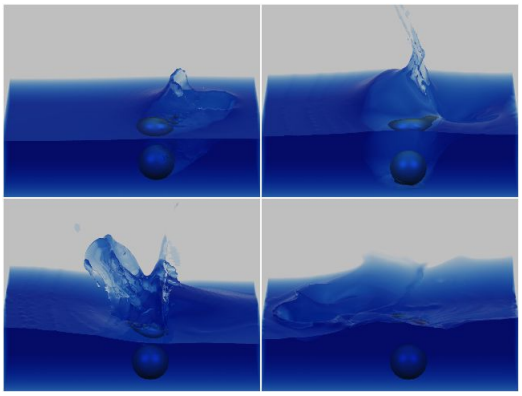

**Figura 3.6:** Practical Animation of Liquids. Nick Foster & Ronald Fedkiw. 2001

métodos basados en elementos finitos. Entre los métodos eulerianos encontramos métodos basados en diferencias finitas, métodos basados en volumenes finitos y métodos de elementos finitos.

Stam [Stam, 1999] reemplazó el sistema de diferencias finitas por un método semi-lagrangiano para la convección de fluidos y un integrador implícito para la difusión a fin de que se pudieran utilizar incrementos de tiempo largos en la animación del humo.

Yngve et al. [Yngve et al., 2000] utilizaron un esquema de diferencias finitas para resolver las ecuaciones para fluidos compresibles de Navier-Stokes en el modelado de ondas de impacto y en la generación de efectos de convección generados por una explosión.

Foster y Fedkiw [Foster and Fedkiw, 2001] revisaron el método de Harlow y Welch y realizaron mejoras como la aproximación semi-lagrangiana para la convección a fin de obtener un modelo más estable dando lugar a un sistema hibrido con partículas lagrangianas y métodos de superficies implícitas *level-set* (LSM, https://en.wikipedia.org/wiki/Level-set_method) para la simulación de fluidos (Figura 3.6). Su método fue mejorado añadiendo otra capa de partículas fuera del agua, técnica denominada *Particle Level Set*, que fue utilizada por Enright et al. [Enright et al., 2002] para crear flujos libres en la superficie con delgadas láminas de salpicaduras y una mejor preservación del volumen del fluido.

Por otra parte, la simulación de un fluido incluye diferentes partes en las que puede ser estudiado: advección, modelado de la superfice, proyección de la presión e interacción entre fluidos y sólidos. Algunos de los trabajos científicos, que citaremos a continuación, se han centrado en alguna de estas partes de la simulación.

Se entiende por advección al transporte horizontal de un fluido (variación de un escalar en un punto dado por efecto de un campo vectorial). Como ejemplos de este proceso de distribución espacial podemos citar el transporte de una sustancia contaminante por la corriente de un río, el transporte de ciertas propiedades, como la salinidad, por las corrientes marinas, etc... Algunas de las aportaciones en esta fase son los trabajos de Fedkiw et al. [Fedkiw et al., 2001], Enright et al. [Enright et al., 2005] y Selle et al. [Selle et al., 2005, Selle et al., 2008].

En cuanto al modelado de la superficie de un líquido se han utilizado métodos de *level-set* [Osher and Sethian, 1988], combinados con partículas [Enright et al., 2002], mallas de triángulos [Bargteil et al., 2006, Müller, 2009, Brochu and Bridson, 2009, Wojtan et al., 2010], *octrees* [Losasso et al., 2004], mallas de tetraedros [Feldman et al., 2005, Klingner et al., 2006, Chentanez et al., 2006, Batty et al., 2007, Batty et al., 2010], mallas de celdas de Voronoi [Sin et al., 2009, Brochu et al., 2010] o mallas de celdas «*tall cell*» [Irving et al., 2006].

En cuanto a la proyección de la presión se utilizan métodos de fluido «fantasma» (*ghost fluid method*) [Enright et al., 2003], métodos del gradiente conjugado precondicionado (PCG) [Foster and Fedkiw, 2001, Bridson, 2008], métodos *multigrid* [Molemaker et al., 2008, Lentine et al., 2010] o hibrídos entre estas dos últimas aproximaciones [Losasso et al., 2004, McAdams et al., 2010].

Con respecto a la interacción entre fluidos y sólidos, se han propuesto volumenes con métodos de fracción del fluido [Takahashi et al., 2002], cuerpos rígidos y la utilización de ocupación de celdas [Carlson et al., 2004, Klingner et al., 2006], *soft body-fluid coupling* [Chentanez et al., 2006, Valkov et al., 2015], *two-way coupling* [Guendelman et al., 2005, Robinson-Mosher et al., 2008] o formulación variacional [Batty et al., 2007].

Otras técnicas que se han utilizado en el modelado y visualización de fluidos son los modelos de ondas [Ts'o and Barsky, 1987, Tessendorf, 1999, Hinsinger et al., 2002, Gamito and Musgrave, 2002], los métodos espectrales [Mastin et al., 1987, Tessendorf, 1999, PremÅŸoe and Ashikhmin, 2000], esquemas eficientes de difusión local [Gomez, 2000, Stam, 2003], sistemas de partículas [Müller et al., 2003, PremÅŸoe et al., 2003, Adams et al., 2007, Solenthaler and Pajarola, 2009], SPH (*Smoothed particle hydrodynamics*) [Lee and Han, 2010], modelos *lattice-Boltzmann* [Thürey and Rüde, 2004], ecuaciones *shallow water* [Thurey et al., 2007, Brodtkorb et al., 2010, Fujisawa et al., 2017], mapas de alturas [Kass and Miller, 1990, Chen and da Vitoria Lobo, 1995], *fluxed animated boundary [Stomakhin and Selle, 2017]* y técnicas de *deep learning* como redes neuronales convolucionales (*Convolutional neural network, CNN*) [Tompson et al., 2016] entre otros.

Para una revisión bibliográfica más completa sobre el modelado y renderizado de fluidos en el área de los gráficos por computador pueden consultarse [Iglesias, 2004, Bridson and Müller-Fischer, 2007, Crane et al., 2007].

En cuanto a los paquetes de software que podemos encontrar en el mercado podemos destacar: *Realflow*, *Flowline*, las herramientas *ANSYS-CFX* y *ANSYS-Fluent*, *OpenFOAM*, *COMSOL Multiphysics* y otros.

*Realflow* (http://www.nextlimit.com/realflow/) es un software para la simulación de fluidos 3D desarrollado por *Next Limit Technologies* (http://www.nextlimit.com/) que utiliza sistemas de partículas y superficies implícitas para el renderizado de fluidos de forma realista.

*Flowline* (http://www.flowlines.info/) es una herramienta interna creada por *Scanline* para crear simulaciones de fluidos realistas que ha sido integrada en la línea de producción del estudio *Moving Picture Company* (http://moving-picture.com/).

La empresa *Ansys* (http://es.wikipedia.org/wiki/ANSYS) incluye herramientas para la simulación de fluidos tales como ANSYS-CFX y ANSYS-Fluent (http://www.ansys.com/es-es/products/fluids) para simular una amplia gama de fenómenos tales como aerodinámica, combustión, hidrodinámica, mezclas de líquidos/sólidos/gas, dispersiones de partículas, flujos de reacción, transferencia de calor, etc.

OpenFOAM (http://www.openfoam.com/products/) es un software gratuito y de código abierto, lanzado y desarrollado principalmente por *OpenFoam Ltd* desde 2004, siendo distribuido más tarde por la *Fundación OpenFoam*, que aborda la resolución de problemas de dinámica de flujos de fluidos que involucran reacciones químicas, turbulencias y transferencia de calor, hasta acústica, mecánica sólida y electromagnética.

*COMSOL Multiphysics* (http://www.comsol.com/comsol-multiphysics), antes conocido como FEMLAB, es un paquete de software de análisis y resolución por elementos finitos para varias aplicaciones físicas y de ingeniería, especialmente fenómenos acoplados, o multifísicos (http://www.comsol.com/models/fluid).

# 4. CAPÍTULO

## Diseño

Como hemos comentado en el capítulo anterior, el proyecto tiene como objetivo la creación de una escena que se simule el comportamiento del agua de forma realista y, posteriormente, muestre los resultados mediante una animación.

En este capítulo presentaremos el modelo que hemos elegido para este fin, pero antes de su desarrollo introduciremos unas definiciones básicas para un mejor entendimiento del mismo, de la definición de las ecuaciones que gobiernan los fluidos en el mundo real, de cómo pueden ser aproximadas numéricamente en ordenadores y del diseño de la aplicación.

En física, un fluido es una sustancia, entre cuyas moléculas solo hay una fuerza de atracción débil, que continuamente se deforma o fluye bajo la aplicación de una tensión tangencial, por muy pequeña que esta sea sin que aparezcan en su seno fuerzas restitutivas que tiendan a recuperar la forma «original». Este término engloba a los líquidos y a los gases siendo los segundos mucho menos viscosos (casi fluidos ideales) que los primeros. Los líquidos toman la forma del recipiente que los aloja, manteniendo su propio volumen, mientras que los gases carecen tanto de volumen como de forma propias. Las moléculas no cohesionadas se deslizan en los líquidos y se mueven con libertad en los gases.

Existen dos enfoques a la hora de analizar el movimiento de un fluido: el punto de vista lagrangiano y el punto de vista euleriano.

Según el primero, un fluido consta de infinitos puntos en el espacio donde cada uno de estos puntos en el fluido se etiqueta como una partícula con posición $\overrightarrow{x}$ y velocidad $\overrightarrow{u}$.

Numéricamente corresponde a un sistema de partículas con o sin una malla asociada que conecta todas las partículas.

Por el contrario, según el punto de vista euleriano, se mantienen puntos fijos en el espacio y se observan la velocidad, la temperatura y otras medidas sobre los fluidos en dichos puntos a lo largo del tiempo. Númericamente corresponde a una red fija que no varía espacialmente cuando el fluido fluye a través de dicha red. La utilización de este modelo permite analizar más fácilmente las derivadas espaciales y también aproximar numéricamente estas derivadas en una malla euleriana fija.

En este caso, se va a tener en cuenta el punto de vista euleriano; es decir, el modelo es una red donde se mantienen los puntos fijos en el espacio y se observa cómo las distintas propiedades del agua varían con el paso del tiempo en dichos puntos.

## 4.1. Definiciones básicas

A continuación, presentaremos algunos conceptos necesarios para el desarrollo de este proyecto.

- *Escena*: Es el espacio en el que se encuentran todos los objetos que se quieren representar y visualizar posteriormente gracias a las cámaras y las luces.
- *Texel*: Es la unidad mínima de una textura aplicada a una superficie.
- *Textura*: En gráficos por computador, un mapa de alturas o un campo de alturas es una imagen de píxeles o imagen en mapa de bits (*bitmap*) que se usa para almacenar valores en forma de matriz, como datos de elevación de superficies, a fin de mostrarlos mediante gráficos 3D. Un mapa de alturas contiene un canal de color que es interpretado como el desplazamiento o la altura desde el "suelo" de una superficie. También puede ser visto como una imagen en escala de grises donde el negro representa la posición más baja y el blanco la mayor altura posible.

  Las dimensiones del mapa de alturas pueden ser de distinto tamaño, y pueden no ajustarse a la malla de representación. Existen dos técnicas que permiten ajustar el mapa de alturas a la malla en función a si el tamaño es mayor o menor.

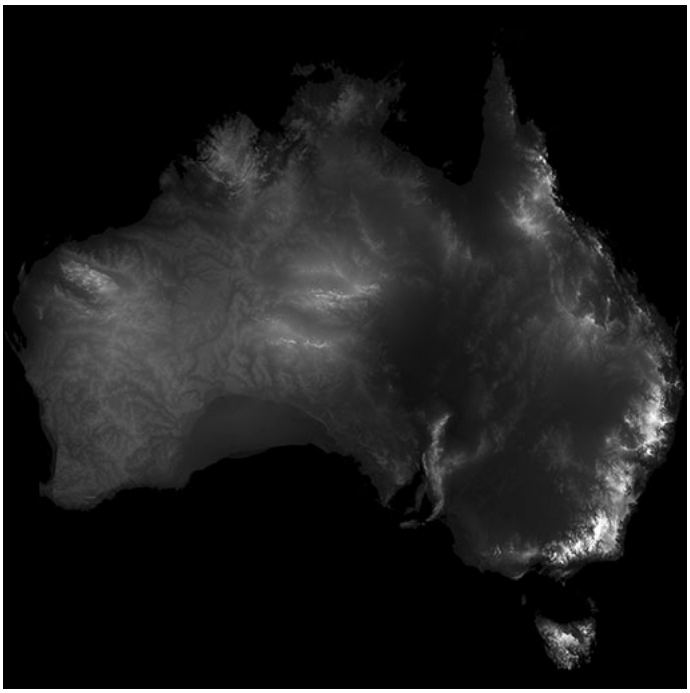

**Figura 4.1:** Mapa de alturas de Australia almacenado como una textura.

- Magnificación: Cuando la imagen de la textura es menor que la superficie que ocupa el polígono en la pantalla se verán los píxeles incrementados. Existen dos formas de solucionar este problema.
  - Interpolación bilineal: Consiste en realizar una interpolación entre los texels más cercanos.
  - Vecino más cercano: Se utiliza el texel más cercano.
- Minificación: Cuando la imagen de la textura es mayor que la superficie que ocupa el polígono en pantalla y más de un texel corresponden a un nodo. El método más conocido para solucionar este problema es el "*mipmapping*".
  - Mipmapping: Se realiza una "pirámide" de la imagen donde en el nivel 0 tenemos la propia imagen. La imagen del nivel $i+1$ tiene la mitad del tamaño del nivel $i$ y los texel del nuevo nivel se obtienen mediante la combinación lineal de 4 texels del nivel anterior.

- *Malla*: Una malla de polígonos es una colección de vértices, aristas y caras que forman una superficie.

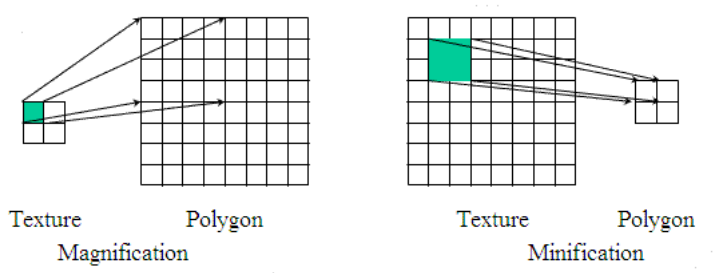


**Figura 4.2:** Ejemplo visual de magnificación y minificación.

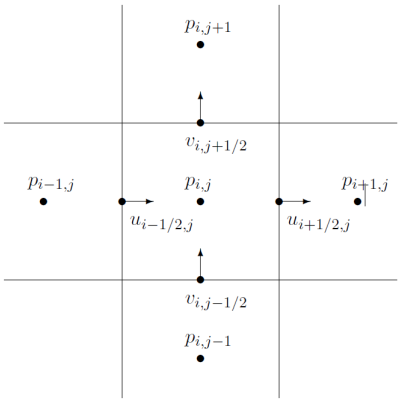


**Figura 4.3:** Esquema de una malla mostrando el punto (i,j) y sus vecinos.

  - Vértice: Un vértice es el punto donde se encuentran dos o más elementos unidimensionales (curvas, vectores, rectas, semirrectas o segmentos).
  - Arista: Segmento donde se encuentran dos caras.
  - Cara: Cada uno de los polígonos que forman o limitan un poliedro.

- *Nodo*: Es un punto de intersección o unión de varios elementos que confluyen en el mismo lugar. En este caso, cada nodo cuenta con distinta información como la posición y la aceleración.
- *Vecino*: Definimos como vecino de un vértice $(i, j)$ en una malla a aquel vértice que comparte una de las coordenadas de posición y donde el valor absoluto de la diferencia entre la otra coordenada es igual a la menor unidad de distancia existente no nula.
- *Vecindario*: El vecindario de un vértice $(i, j)$ es el conjunto formado por los vértices vecinos de este.

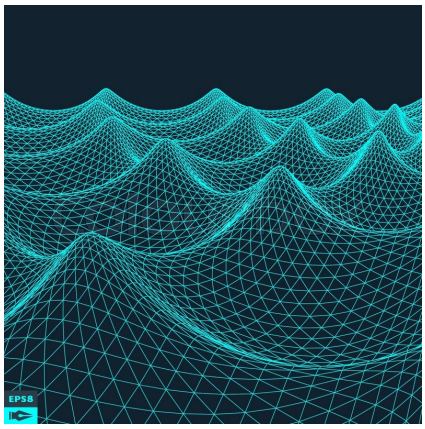

**Figura 4.4:** Malla cuyos nodos se encuentran conectados.

## 4.2. Dinámica de fluidos

La dinámica es la parte de la física que estudia el movimiento en relación con las causas que lo producen. La dinámica de fluidos estudia los fluidos en movimiento y las interacciones entre el fluido y el contorno que lo limita.

Un fluido es cualquier sustancia que fluye y carece de rigidez y elasticidad, y en consecuencia cede inmediatamente a cualquier fuerza tendente a alterar su forma y adoptando así la forma del recipiente que lo contiene.

Los cuerpos rígidos tienen forma y orientación además de posición, masa y velocidad. Si se agrega la noción de forma a una partícula, se obtiene un cuerpo rígido. Los cuerpos deformables pueden cambiar de forma pero conservan su conectividad entre varios puntos del mismo. Los fluidos tienen mucha libertad de movimiento y éste no es lineal. Además, su forma y topología pueden cambiar. Los fluidos requieren técnicas de simulación especializadas debido a que toman la forma de su contenedor y siempre están en colisión con todo lo que los rodea.

Para realizar la simulación, inicialmente se debe de crear una escena, en la cual, el elemento más importante es la malla que simulará la superficie de agua que se quiere representar. Esta malla, está compuesta por nodos, los cuales se encuentran en cada vértice que forma la geometría de la malla. Los nodos cuentan con información necesaria para la simulación:

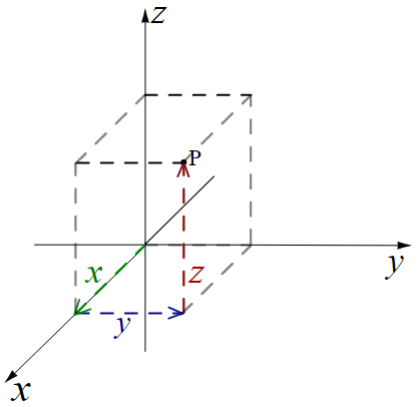


**Figura 4.5:** Representación de la posición del punto *P* en el sistema de coordenadas cartesianas.

- *Posición*: La posición de una partícula indica su localización en el espacio y se representa como una magnitud vectorial respecto a un sistema de coordenadas de referencia.

- *Masa*: Es una magnitud que expresa la cantidad de materia de un cuerpo, medida por la inercia de este, que determina la aceleración producida por una fuerza que actúa sobre él.

- *Viscosidad*: Propiedad física característica de todos los fluidos, la cual emerge de las colisiones entre las partículas del fluido que se mueven a diferentes velocidades, provocando una resistencia a su movimiento.

- *Velocidad*: Es una magnitud física de carácter vectorial que indica un desplazamiento de un punto material en un tiempo dado y en una dirección dada. La velocidad siempre está definida por un sistema de referencia.

- *Aceleración*: Es una magnitud derivada vectorial que indica un cambio de velocidad en el tiempo.

- *Fuerza*: En física, la fuerza es una magnitud vectorial que mide la razón del cambio del momento lineal entre dos partículas o sistemas de partículas.

## 4.3. Ecuaciones de Navier-Stokes

En la actualidad, alguien que quiere realizar una animación de líquidos está prácticamente obligado a conocer qué son las ecuaciones de Navier-Stokes. Estas ecuaciones reciben su nombre de Claude-Louis Navier y George Gabriel Stokes. Este par de ecuaciones son derivadas parciales no lineales que describen el movimiento de un fluido. Estas ecuaciones gobiernan la atmósfera terrestre, las corrientes oceánicas y el flujo alrededor de vehículos o proyectiles y, en general, cualquier fenómeno para todo tipo de fluidos.

Las ecuaciones de Navier-Stokes vienen dadas por:

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} + \frac{1}{\rho} \nabla p = \vec{g} + \nu \nabla \cdot \nabla \vec{u} \tag{4.1}$$

$$\nabla \cdot \vec{u} = 0 \tag{4.2}$$

donde $\vec{u}$ corresponde a la *velocidad del fluido*, $\nabla$ es el operador gradiente[1], $t$ indica la *unidad de tiempo*, $\rho$ hace referencia a la *densidad del fluido* que relaciona la masa y el volumen del fluido, $p$ denota la *presión* que ejerce el líquido, $\vec{g}$ representa la *gravedad* $(0, -9{,}81, 0) m/s^2$ y $\nu$ corresponde a la *viscosidad cinemática* que es una propiedad de los fluidos que indica la mayor o menor resistencia que estos ofrecen al movimiento de sus partículas cuando son sometidos a una fuerza tangencial ().

La primera ecuación (4.1) surge a partir de la segunda ley de Newton[2] y combina los campos de velocidad y presión y las relaciona mediante la conservación del momento. Esta segunda ley de Newton indica que el cambio de movimiento o aceleración es directamente proporcional a la fuerza motriz impresa e inversamente proporiconal a la masa del cuerpo.

$$a = \frac{\sum \vec{F}}{m} \tag{4.3}$$

A partir de esta, podemos entender cómo la viscosidad, la gravedad o la presión están vinculadas a la aceleración y cómo estas fuerzas, suponiendo la masa de cada unidad de fluido es de valor 1kg, equivale a la aceleración que se ejerce sobre cada punto del fluido.

[1] El operador gradiente viene dado por las derivadas parciales: $\nabla = (\partial/\partial x, \partial/\partial y, \partial/\partial z)$

[2] Esta ley expresa que la aceleración que adquiere un cuerpo es proporcional a la fuerza neta aplicada sobre el mismo y la constante de proporcionalidad es la masa del cuerpo. En resumen, esta ecuación se encarga de cuantificar el concepto de fuerza.

La segunda ecuación (4.2) es la denominada “ecuación de incompresibilidad”. El estudio de flujo compresible es costoso computacionalmente, por ello se trabaja con este tipo de ecuaciones. Para ello se tiene que cumplir la condición de que el gradiente de la velocidad del fluido sea igual a 0. Esta ecuación indica que la masa siempre debe conservarse (incompresibilidad del fluido).

Los fluidos como puede ser el agua, cambian su volumen, pero generalmente la variación no es notoria. Para un fluido incompresible, el volumen debe de mantenerse constante, por lo que la velocidad de cambio debe de ser cero.

$$d\ dt\ volumen(\Omega) = \int\int \partial\Omega \vec{u} \cdot n = 0 \tag{4.4}$$

$$d\ dt\ volumen(\Omega) = \int\int\int \Omega \nabla \cdot \vec{u} = 0 \tag{4.5}$$

Finalmente, esto nos lleva a la ecuación de incompresibilidad descrita previamente (4.2).

Si aplicamos a cada nodo de la malla, a partir de un estado inicial $t$, las ecuaciones de Navier-Stokes obtenemos el nuevo estado de la malla que corresponde al nuevo estado del fluido en el tiempo $t+1$.

## 4.4. Modelo elegido: The Height Field Model

En este proyecto hemos utilizado un mapa de alturas para modelar la superficie del agua. Esto significa que la altura de una superficie se modela como una función de las coordenadas espaciales $x$ e $y$. Esta representación reduce el tamaño del problema en una dimensión.

Las limitaciones de la representación de un volumen de agua mediante un campo de alturas pueden ser notorias por el siguiente motivo. Es posible una representación de olas, por ejemplo, de una superficie oceánica o un estanque. Sin embargo, las salpicaduras y las olas rompientes no pueden ser representadas mediante este modelo. A pesar de todo, es una representación útil y la utilización de un campo de alturas será el enfoque de este proyecto.

Como consecuencia de esta reducción dimensional, contaremos con un mejor rendimiento debido a la reducción de cálculos a realizar. Un sistema de partículas que represente un volumen de agua o cualquier fluido, depende de las tres coordenadas espaciales ($x$, $y$, $z$), ya que cada partícula es independiente al resto y puede moverse por todo el espacio

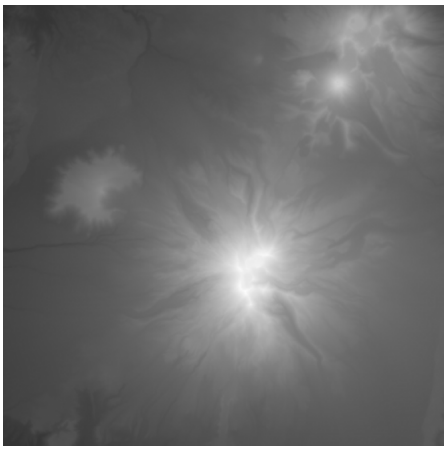

**Figura 4.6:** Mapa de alturas de Mt Ruapehu y Mt Ngauruhoe (Nueva Zelanda)

libremente. En cambio, un mapa de alturas cuenta con dos coordenadas espaciales fijas, las cuales determinan la posición de cada nodo y una variable que determina la altura del fluido en ese punto.

Por otra parte, mientras que con un sistema de partículas, el volumen representado influye en la cantidad de cálculos a realizar debido a la necesidad de añadir una cantidad mayor de éstas y por lo tanto un descenso de frames renderizados por segundo, en el caso de un mapa de alturas, si el espacio no varía en las coordenadas $x$ e $y$, el coste computacional sigue siendo exactamente el mismo. En el caso de ampliar alguna de estas dos dimensiones, simplemente se añadiría una fila o columna de nodos que es menor al número de partículas que corresponden a ese volumen aditivo.

Recordando lo descrito previamente, la malla va a ser la encargada de mostrar la simulación calculada. Los valores iniciales de esta malla son los correspondientes al mapa de alturas inicial.

Este mapa de alturas corresponde a una textura de escala de grises, donde el color blanco se puede interpretar como el punto de mayor altura representable y el negro el punto de menor altura representable. En una imagen de 8 bits, únicamente se pueden representar 256 valores. En función al tamaño de la imagen, este intervalo puede variar.

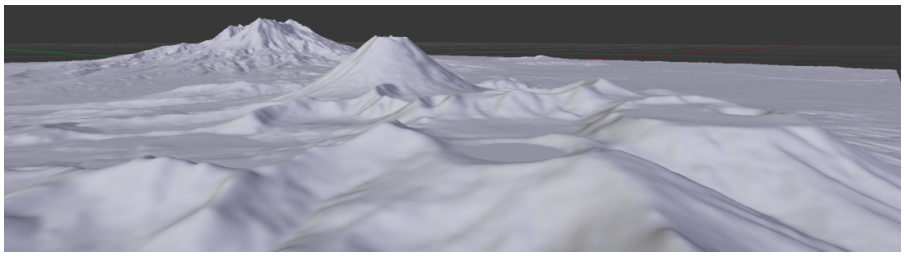

**Figura 4.7:** Terreno obtenido a partir del mapa de alturas de Mt Ruapehu y Mt Ngauruhoe (Nueva Zelanda)

### 4.4.1. Cálculo del gradiente en un mapa de alturas

El gradiente es la variación de una magnitud en función de la distancia. Para calcular esta variación en un mapa de alturas, dado un nodo $(i, j)$ se toman el vecindario como elementos a tener en cuenta para calcularlo.

Tomando como referencia la figura de la definición de vecindario, podemos decir que el gradiente del elemento $(i, j)$ corresponde a la variación que existe con cada uno de sus vecinos dando como resultado la siguiente expresión:

$$\nabla f(i,j) = h(i+1,j) + h(i-1,j) + h(i,j+1) + h(i,j-1) - 4 * h(i,j) \tag{4.6}$$

## 4.5. Métodos de Integración

La integración numérica consta de una amplia gama de algoritmos para calcular el valor numérico de una integral definida y, por extensión, el término se usa a veces para describir algoritmos numéricos para resolver ecuaciones diferenciales.

En este apartado vamos a ver dos métodos comunes en la representación de fluidos que son el método de integración de Euler y el método de integración de Verlet.

### 4.5.1. Método de Euler

El método de Euler es el más simple de los métodos numéricos para resolver un problema de valor inicial, como es este caso, donde contamos con un mapa de alturas inicial. El método de Euler es un método de primer orden, lo que significa que el error local y el

global son proporcionales al cuadrado del tamaño del paso. El método de Euler sirve como base para construir métodos más complejos.

La obtención del valor correspondiente a un nodo de la malla en el instante $t+1$ conociendo el valor del instante $t$ se realiza de la siguiente forma:

$$y_{t+1} = y_t + hf(x_t, y_t) \tag{4.7}$$

donde $y_{t+1}$corresponde al valor de un nodo en el momento $t+1$ (que es el que queremos calcular), $y_t$es el valor correspondiente en el momento $t$, $h$ hace referencia a la variación temporal que existe entre los instantes $t+1$ y $t$, y la función $f(x_t, y_t)$ devuelve como resultado la variación que existe entre los pasos $t+1$ y $t$. Esta última función aplicada al escenario en el que estamos trabajando, corresponde con la variación que existe con su vecindario.

### Error local

El error local corresponde al error cometido en un único paso. Es la diferencia entre la solución numérica tras un paso, $y_1$, y la solución exacta en el momento $t_1 = t_0 + h$. El valor viene dado por:

$$y_1 = y_0 + hf(t_0, y_0) \tag{4.8}$$

La solución exacta viene dada por el desarrollo de Taylor:

$$y(t_0 + h) = y(t_0) + hy'(t_0) + \frac{1}{2}h^2 y"(t_0) + O(h^3) \tag{4.9}$$

El error cometido viene dado por la diferencia entre estas ecuaciones, es decir:

$$y(t_0 + h) - y_1 = \frac{1}{2}h^2 y"(t_0) + O(h^3) \tag{4.10}$$

Como se puede ver, el error cometido es aproximadamente proporcional a $h^2$.

### Error global

El error global es la acumulación de los errores locales cometidos en cada iteración, asumiendo que el valor inicial no varía con el resultado real.

El número de pasos a realizar es $\frac{(t-t_0)}{h}$, que es proporcional a $\frac{1}{h}$, y el error cometido en cada paso es proporcional a $h^2$. Por lo tanto, el error global cometido es proporcional a $h$.

### 4.5.2. Método de Verlet

El algoritmo de Verlet es un procedimiento para la integración numérica de ecuaciones diferenciales ordinarias de segundo orden con valores iniciales conocidos. Es muy utilizado en problemas de dinámica newtoniana, por lo que se emplea frecuentemente en este tipo de problemas.

Dado un sistema de coordenadas, la posición de un nodo a lo largo del tiempo puede ser expresada mediante una función de las coordenadas y del tiempo. Realizando la suma de los desarrollos de la serie de Taylor de $\vec{x}(t+\Delta t)$ y $\vec{x}(t-\Delta t)$ obtenemos que:

$$\vec{x}(t+\Delta t) = 2\vec{x}(t) - x(t-\Delta t) + \vec{a}(t)\vec{\Delta t}^2 + O(\Delta t^4) \tag{4.11}$$

El error cometido es de cuarto grado (despreciando el último término), y teniendo en cuenta que el intervalo de tiempo es menor que la unidad, el error cometido resulta poco significante. En este caso, la ecuación no depende de la velocidad lo cual simplifica los cálculos.

#### Error local

Al igual que en el método de Euler, y a partir de la ecuación obtenida (4.11), el error que se comete es del orden $\Delta t^4$, inferior a la mayoría de los algoritmos como por ejemplo el de Euler.

#### Error global

El número de pasos a realizar es el mismo obtenido que con el método anterior. Esto es, ya que es proporcional a $\frac{1}{\Delta t}$, podemos decir que el error global cometido es proporcional a $\Delta t^3$.

### 4.5.3. Método de Verlet con velocidad

Un algoritmo relacionado y más comúnmente utilizado, es el algoritmo de Verlet con velocidad. Este guarda cierta similitud al método de salto o «leap-frog» salvo porque la velocidad y la posición se calculan con el mismo valor de la variable de tiempo. Utiliza un enfoque similar, pero incorpora explícitamente la velocidad, resolviendo el problema del primer paso del algoritmo de Verlet básico:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + v(\vec{t})\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \tag{4.12}$$

$$v(t \vec{+} \Delta t) = \vec{v}(t) + \frac{\vec{a}(t) + \vec{a}(t+\Delta t)}{2}\Delta t \tag{4.13}$$

El algoritmo es una mejora, sin embargo, el error cometido sigue siendo el mismo que el método básico.

# 5. CAPÍTULO

# Desarrollo del proyecto

## 5.1. Computación GPGPU

GPGPU es un concepto reciente dentro de informática que trata de estudiar y aprovechar las capacidades de cómputo de una GPU.

Según define la Wikipedia, https://es.wikipedia.org/wiki/GPGPU, una GPU es un procesador diseñado para los cómputos implicados en la generación de gráficos 3D interactivos. Algunas de sus características se consideran atractivas para su uso en aplicaciones fuera de los gráficos por computadora, especialmente en el ámbito científico y de simulación. Estas características son el bajo precio en relación a su potencia de cálculo, la capacidad de paralelizar y la optimización para cálculos en coma flotante.

Principalmente, el desarrollo de software GPGPU se ha realizado en lenguaje ensamblador, o en lenguajes específicos para aplicaciones gráficas usando la GPU como son GLSL, Cg o HLSL. Sin embargo, han surgido herramientas para facilitar el desarrollo de aplicaciones GPGPU, al abstraer muchos de los detalles relacionados con los gráficos y presentar una interfaz de más alto nivel como es BrookGPU.

Una herramienta extendida es CUDA, propiedad de Nvidia, la cual es una extensión de C que permite la codificación de algoritmos en GPU de dicha empresa.

Los programas que realizan cálculos en GPGPU son los llamados “shaders”. Estos shaders son utilizados para realizar transformaciones de vértices o coloreado de píxeles. Para este proyecto, vamos a utilizar distintos shaders. El primero tiene como función el uso

que se le da habitualmente a un shader, que es el de representar una escena. El segundo es un shader especial cuya finalidad es de realizar todos los cálculos necesarios para la simulación del proyecto.

Un elemento a destacar en los shaders son las variables “uniform”. La GPU maneja grandes números de threads en paralelo. A pesar de que cada thread no conoce a los otros, se necesita poder enviarle valores de entrada desde la CPU a todos ellos. Todos estos valores van a ser iguales para todos los threads y con la característica de que únicamente son de lectura. Estas variables pueden ser de diferentes tipos entre los que se destacan los números de coma flotante y vectores de 2,3 y 4 dimensiones.

## 5.2. Computación GPGPU y la simulación de fluidos

## 5.3. Algoritmo principal

El algoritmo principal sigue el siguiente esquema:

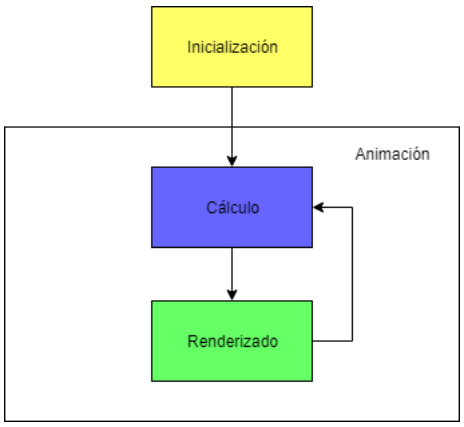


**Figura 5.1:** Esquema del algoritmo principal.

Como se puede observar, el proyecto comienza con la inicialización, donde se preparan todos los elementos para posteriormente realizar la animación. Después se pasa al bucle de renderizado o animación, donde primero se realizan los cálculos que posteriormente se van a mostrar, y finalmente se realiza y muestra el renderizado de la escena. Este bucle está en constante ejecución salvo que se detenga de forma manual.

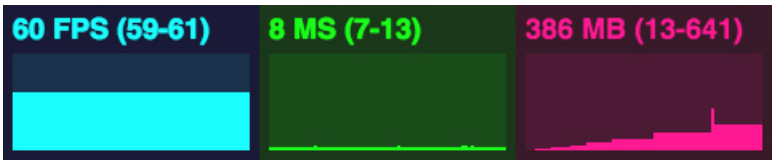


**Figura 5.2:** Las distintas interfaces que muestra el paquete stats.

### 5.3.1. Interfaz con el usuario

Existe la opción de crear un menú desplegable donde se pueden modificar distintas variables como la viscosidad.

En él, también se puede añadir una nueva opción donde se decida qué método de simulación utilizar. De esta forma se podría elegir entre el implementado en un inicio o la nueva implementación del método de Verlet.

Por otra parte, gracias a la librería stat, se puede mostrar el número de frames por segundo al que se está mostrando la animación por pantalla. Si se hace clic en él, también se puede ver más información el número de milisegundos necesarios para renderizar un fotograma (cuanto menor sea el número, mejor) y el número de MegaBytes de memoria que utiliza.

### 5.3.2. Bucle de renderizado

El bucle de renderizado consta de dos partes. La primera parte consiste en el cálculo del estado de la malla en siguiente momento temporal, es decir, si la malla representada es la correspondiente al tiempo $i$, se procede al cálculo del estado de la malla en el tiempo $i+1$. La segunda renderiza la escena actualizada como cualquier programa de animación. Hasta que algún evento lo impida (por ejemplo el de cerrar la simulación), se repetirán estos pasos en este orden y sin detenerse.

```
1 funcion animar():
2     calcularNuevoEstado();
3     renderizar();
4     animar();
5 fin animar;
```

## 5.4. Funciones principales

El proyecto se puede dividir en dos subconjuntos principalmente donde, el primero está relacionado con la representación de la simulación y el segundo se encarga del cálculo de cada nodo.

### 5.4.1. Creación y renderizado de la escena

#### Creación de la escena

El primer paso de ejecución es la creación de la escena que se corresponde a la función init() de cualquier código escrito en Three.js. En este paso se prepara todo lo necesario para posteriormente poder llevar a cabo la simulación.

En primer lugar se crea una cámara en perspectiva y se le asigna una posición donde estará ubicada. Después se crea una nueva escena donde se encontrarán todos los elementos. También será necesario crear la iluminación y añadirla a la escena. El renderer es el elemento que procesa las escenas y que posteriormente se visualizan por pantalla.

Para concluir con la inicialización, se debe inicializar el elemento imprescindible, que es la malla que simulará el agua. Esta función se encarga de crear un plano, aplicarle un material, asignarle distintos valores de las propiedades del material a las uniformes, colocar el objeto en la escena y mostrar por pantalla la escena.

#### Vertex-shader y fragment-shader para el renderizado de la escena

Para el renderizado de la escena, necesitamos un vertex-shader y un fragment-shader. En este caso, se cuenta con un vertex shader llamado “waterVertexShader” en el cual se calcula la normal de cada vértice en función a sus vecinos y se normaliza, también se calculan la posición de visión, y el vértice transformado.

En cuanto al fragment shader, no se cuenta con uno creado desde cero, sino que se utiliza uno predeterminado de Three.js. Mediante la función “THREE.ShaderChunk[ ’meshphong_frag’ ]” cargamos el fragment shader predeterminado.

El material «mesh phong» se caracteriza por sus superfices brillantes con reflejos especulares. Este material utiliza un modelo que no se basa en la física para calcular la reflectancia. El sombreado se calcula usando un modelo de sombreado Phong.

### 5.4.2. Computación GPGPU

La computación GPGPU consiste en el aprovechamiento de la unidad de procesamiento gráfico para realizar los cálculos. De esta forma se aprovecha la cantidad de núcleos de los que dispone para realizar cálculos con mayor rapidez.

#### INICIALIZACION DE GPGPU

La inicialización de GPGPU se realiza después de crear la malla. Dentro de la función "init()" se llama a la función "initWater()". En ella creamos un renderer especial mediante la función "GPUComputationRender", el cual servirá exclusivamente para el cómputo en paralelo gracias a la unidad de procesamiento gráfico (GPU). Se crea una textura inicialmente vacía y posteriormente se «rellena», como indica el nombre de la función, esa textura creada. Esta función se encarga de asignar los valores al mapa de alturas (la textura creada) los valores de forma personalizada donde, la componente $x$ será el valor de la altura del agua en cada nodo, el valor $y$ y $z$ se les asigna el valor 0 y al canal alpha el valor 1. Cabe destacar que pese a asignar el valor 0 a la componente $y$, posteriormente almacenará información que, si corresponde al método de Euler será la velocidad del fluido en ese nodo y si corresponde al método de Verlet será la posición del nodo en el tiempo $t-1$.

Después se definen las dependencias entre variables y, en este caso, la dependencia de la variable del mapa de alturas es con ella misma.

Se asignan los valores de las uniformes como son la posición y el tamaño del ratón, la constante de viscosidad y los límites del heightmap y se inicia la computación donde, en caso de error, se notifica por la consola del navegador. Una variable uniforme en GLSL es una variable global que tienen este nombre porque no se modifican en la ejecución del shader.

Tras estos pasos se finaliza la inicialización de la GPGPU. A continuación se pasa al bucle del programa donde, como previamente se ha dicho, se realizan los cálculos del nuevo estado de la malla para el siguiente paso y se renderiza la malla en el nuevo estado tomando como valores iniciales los calculados con anterioridad.

### Fragment-shader para la computación GPGPU

El fragment shader es el "programa" encargado de calcular los nuevos valores de cada nodo de la malla. Este cálculo corresponde al siguiente pseudocódigo para cada nodo, siendo un nodo$(i, j)$ y usando el método de Euler:

```
suma_parcial = suma_altura_vecinos - 4*nodo(i,j);
aceleracion = suma_parcial * gravedad + suma_parcial * viscosidad;
velocidad = velocidad + aceleracion;
altura = altura + velocidad * intervalo_tiempo;
```

Tras el cálculo del gradiente y la aceleración, la aplicación del método semi-implícito de Euler refleja el siguiente esquema:

$$\vec{y_{t+1}} = \vec{y_t} + \vec{a_t} * dt \tag{5.1}$$

$$x_{t+1} = x_t + \vec{y_{t+1}} * dt \tag{5.2}$$

Lo que se realiza en un inicio es el cálculo de la variación entre el nodo que se está calculando y sus vecinos. Esa variación multiplicada por la gravedad da como resultado la aceleración. A esta aceleración se le debe de añadir el factor de viscosidad como otra componente dentro de la suma de fuerzas que intervienen en el fluido. En verdad, la suma de la viscosidad y la gravedad no es más que una suma de fuerzas. Sin embargo, recordemos que, al igual que se dijo en el diseño, vamos a considerar que todos los nodos de agua tienen masa 1kg lo cual numéricamente hace que el valor del sumatorio de fuerzas sea equivalente a la aceleración. Sumando la aceleración a la velocidad del paso previo se obtiene la nueva velocidad. Con el mismo criterio, se suma a la altura la velocidad multiplicada por el intervalo de tiempo que se está tomando para cada paso.

Por último también se tiene en cuenta la influencia que tiene el ratón en la escena para alterar la altura en esa zona en función a su posición y el diámetro establecido en el menú desplegable que se encuentra en la parte superior derecha de la interfaz.

En el caso de Verlet, el pseudocódigo para cada nodo $(i, j)$ sería el siguiente:

```
suma_parcial = suma_altura_vecinos - 4*nodo(i,j);
aceleracion = suma_parcial * gravedad + suma_parcial * viscosidad;
altura = 2*altura - altura_anterior + aceleracion * invervalo_tiempo^2;
```

Tras el cálculo de la aceleración en el instante actual al igual que en Euler, el método de Verlet de posición sigue el siguiente esquema:

$$x_{t+1} = 2 * x_t - x_{t-1} + \vec{a_n} * (\Delta t)^2 \tag{5.3}$$

A diferencia del método de Euler, el algoritmo de Verlet es una recurrencia de segundo orden donde, para calcular el estado del instante $t+1$, necesitas conocer el estado en los instantes $t$ y $t-1$. Además, para calcular el estado en el instante $t+1$, no es necesario conocer la velocidad, únicamente la aceleración.

Si comparamos ambos algoritmos a nivel de memoria, ambos utilizan la misma cantidad de espacio.

Por una parte, el método de Euler depende de la velocidad, que a la hora de la implementación, se almacena en la componente *y* del mapa de alturas. En el método de Verlet, la velocidad no es necesaria para calcular la altura de un nodo en cualquier momento temporal.

Por otra parte, al ser el método de Verlet un algoritmo de recursividad de segundo orden, es necesario un lugar en memoria para guardar las alturas de cada nodo de la malla en un paso previo más. Este espacio puede ser el de la componente *y* del mapa de alturas que hemos librado al ser innecesaria la velocidad.

Como mejor del algoritmo de Verlet básico, se encuentra el algoritmo de Verlet con velocidad con el siguiente pseudocódigo:

```
altura = altura + velocidad * intervalo_tiempo + 1/2 * (aceleracion * intervalo_tiempo^2);
nueva_aceleracion = suma_parcial * gravedad + suma_parcial * viscosidad;
aceleración_media = (aceleracion + nueva_aceleracion)/2;
velocidad = velocidad + aceleración_media * intervalo_tiempo;
aceleracion = nueva_aceleracion;
```

Tras el cálculo de la aceleración como en los dos métodos previos, el método de Verlet con velocidad sigue el siguiente esquema:

$$x_{t+1} = x_t + y_t * \Delta t + \frac{1}{2} * a_t * (\Delta t)^2 \tag{5.4}$$

$$a_{t+1/2} = (\vec{a_t} + a_{t+1})/2 \tag{5.5}$$

$$y_{t+1} = y_t + a_{t+1/2} * \Delta t \tag{5.6}$$

Como motivo de reducir el coste computacional, se va a considerar que la viscosidad del fluido será una constante. Dicha constante podrá ser modificada durante el transcurso de la ejecución. Sin embargo, la física no define la viscosidad cinemática como una constante sino que ésta se ve alterada por distintos factores en cada momento. Es la relación que existe entre la viscosidad dinámica y la densidad.

También existe otra función la cual se llama «smoothWater» la cual reduce el ruido del mapa de alturas en ese instante. Debido a que debe de manipular el mapa de alturas y esta modificación es más eficiente en paralelo, cuenta con otro fragment shader para realizar el cálculo. Para dicha reducción, para cada nodo realiza la media entre su altura y la de su vecindario.

## 5.5. Librerías utilizadas

A continuación se explican las librerías utilizadas en este proyecto:

1. `Three.js`: Es una biblioteca que sirve para crear y mostrar animaciones por ordenador en 3D en un navegador web mediante JavaScript. Mediante ella, se importa todo lo necesario para representar una escena. Cuenta con algunos elementos predefinidos como por ejemplo los materiales, y más de 150 archivos de códigos de ejemplo más las fuentes, modelos, textuars y otros elementos.

2. `Detector.js`: Librería cuya función principal es detectar si WebGL es compatible con el navegador en el que se está ejecutando el programa.

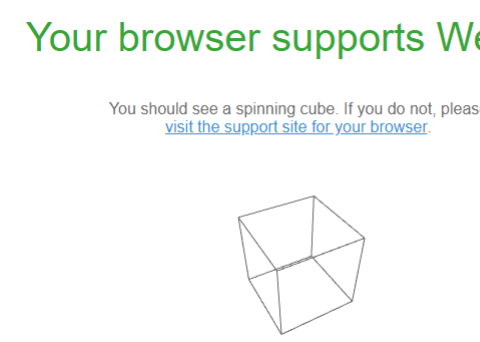


**Figura 5.3:** Página web de WebGL indicando que el buscador soporta WebGL.

3. `stats.min.js`: Es la librería encargada de mostrar información de rendimiento acerca de la animación que se está ejecutando. En esa información se encuentra el número de fotogramas que se muestran por segundo y la memoria en megabyes que se está utilizando, el tiempo necesario para renderizar un fotograma y también la opción de añadir más paneles con información personalizable.

4. `dat.gui.min.js`: Es una librería ligera que permite crear interfaces de usuario que permiten modificar variables en JavaScript. En este caso nos permitirá interactuar con la escena en el tiempo de ejecución. Por ejemplo, modificando la viscosidad del fluido o el tamaño del ratón.

5. `OrbitControls.js`: Permite a la cámara orbitar alrededor del objetivo, en este caso, la malla de representación. Permite la rotación automática, la rotación por teclado, hacer zoom o rotación por ratón entre otros.

6. `SimplexNoise.js`: Librería cuyas funciones principalmente añaden ruido a una imagen.

7. `GPUComputationRenderer.js`: Es la librería que permite la computación en la unidad de procesamiento gráfico y gracias a ello la simulación en tiempo real.

# 6. CAPÍTULO

# Análisis de los resultados

Cuando se realiza el análisis de los métodos numéricos utilizados se tienen en cuenta la validez del modelo en función de sus propiedades de consistencia, estabilidad, convergencia y precisión. Se dice que un modelo es consistente cuando la ecuación discretizada tiende a la ecuación diferencial cuando el paso tiende a cero (error de truncamiento). Un modelo es estable si los errores no aumentan indefinidamente al progresar en el cálculo de un paso de tiempo a otro. Un modelo converge cuando la solución numérica tiende a la solución exacta de la ecuación diferencial cuando el paso tiende a cero.

En este capítulo se presentan los resultados obtenidos tras la ejecución del programa incluidos los problemas encontrados durante su implementación.

Por otra parte, presentaremos las diferencias encontradas en la visualización de la simulación del fluido para los diferentes esquemas numéricos que se han implementado en este trabajo.

## 6.1. Problemas encontrados en la fase de implementación

En cuanto a los problemas que hemos encontrado en la fase de implementación, podemos citar algunos inconvenientes leves como los errores cometidos al realizar el producto entre un número decimal y uno entero (no permitido por GLSL) y otros que han supuesto un mayor problema. En este apartado se van a hablar acerca de estos últimos.

- Inestabilidad e imprecisión en las simulaciones: Tras realizar la implementación de los tres métodos de integración para la simulación de fluidos más conocidos, se realizaron pruebas y se detectaron simulaciones inestables y, algunas, que presentaban efectos imprecisos y, quizás, «anómales». Por ello, se ha realizado un análisis visual de las ejecuciones y se puede observar, a simple vista, que las simulaciones no han sido satisfactorias debido a la gran diferencia existentes entre todas ellas. En el siguiente punto, se analizan los tres métodos implementados y estudiados en esta memoria y sus resultados.

- Convergencia inexacta: El cálculo de la escena es exacto despreciando cualquiera de las fuerzas que deceleran la escena. Sin embargo, debido al hecho de haber simplificado algunos factores y haber omitido otros, la ejecución puede no finalizar en un escenario donde el fluido se encuentre en estado de reposo. Como solución alternativa, en caso de querer aproximarse a ese resultado, se puede aplicar a las aceleraciones o velocidades un factor que reduzca su valor. Sin embargo, esta solución no forma parte de las ecuaciones de fluidos ni de los métodos, sólamente sería para conseguir un efecto.

## 6.2. Resultados obtenidos

A continuación, se muestran los resultados obtenidos tras la implementación de los tres métodos de integración realizados.

### 6.2.1. Método de Euler

El método de Euler se basa en la obtención del nuevo estado del mapa de alturas calculando la velocidad que se ve alterada en función a la aceleración en cada momento de cálculo.

En primera instancia, este método debe de ser el que mayor error exponga en su resultado. Tras realizar varias ejecuciones de larga duración, se ha podido comprobar que el método propuesto por Euler es estable, es decir, no se descontrola. Por otra parte, se puede decir que el resultado es convergente. Este hecho es fácilmente apreciable visualmente, donde se puede verse ese efecto característico del agua y sus ondas con facilidad.

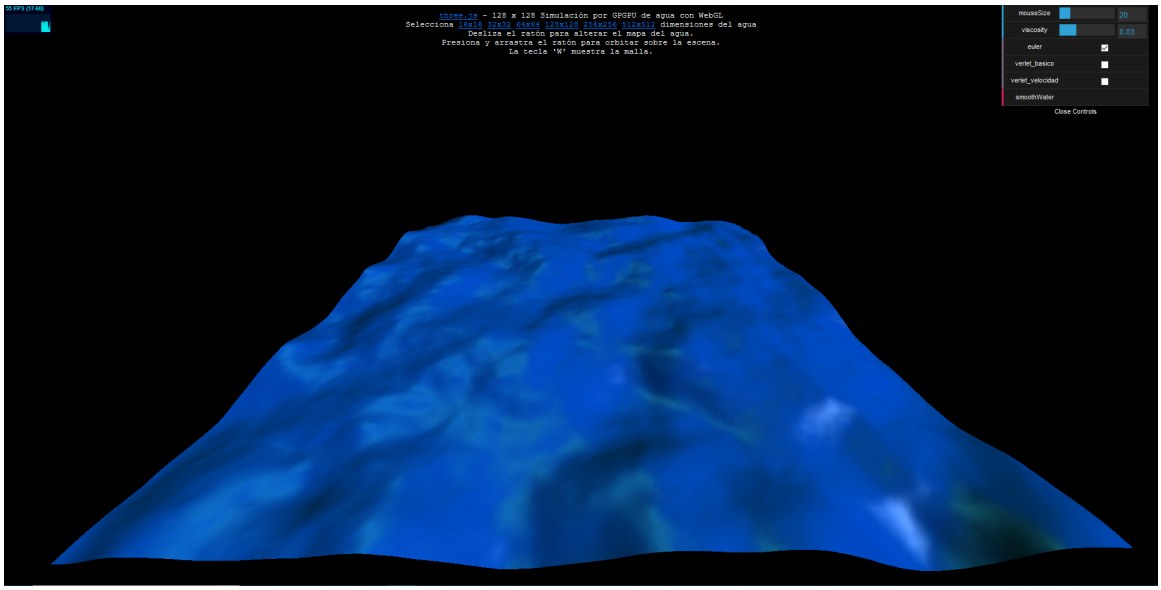


**Figura 6.1:** Captura de la ejecución aplicando el método de Euler.

Otro de los elementos a tener en cuenta es la baja influencia que tiene el factor de viscosidad. Es dificilmente apreciable su participación en ejecución dentro del cálculo de fuerzas.

### 6.2.2. Método de Verlet

El método de Verlet consiste en la obtención del nuevo estado del mapa de alturas gracias al conocimiento del estado de este mismo en los dos pasos anteriores.

Este método de integración parece ser más preciso que el de Euler y el error cometido es de un orden mayor y, por lo tanto, mejor. Tras varias ejecuciones se ha podido comprobar que el resultado no es tan satisfactorio como lo esperado. La simulación carece de fluidez y parece estar lejos de asemejarse al agua. Este algoritmo es estable y convergente. Sin embargo, pese a omitir las fuerzas que afectan al fluido como es la viscosidad (asignándole al coeficiente de viscosidad el valor 0.0), el resultado obtenido comparado con el método de Euler varía considerablemente.

Al igual que el anterior método, es destacable la baja influencia que tiene el factor de viscosidad a la hora de calcular las fuerzas que intervienen en la simulación.

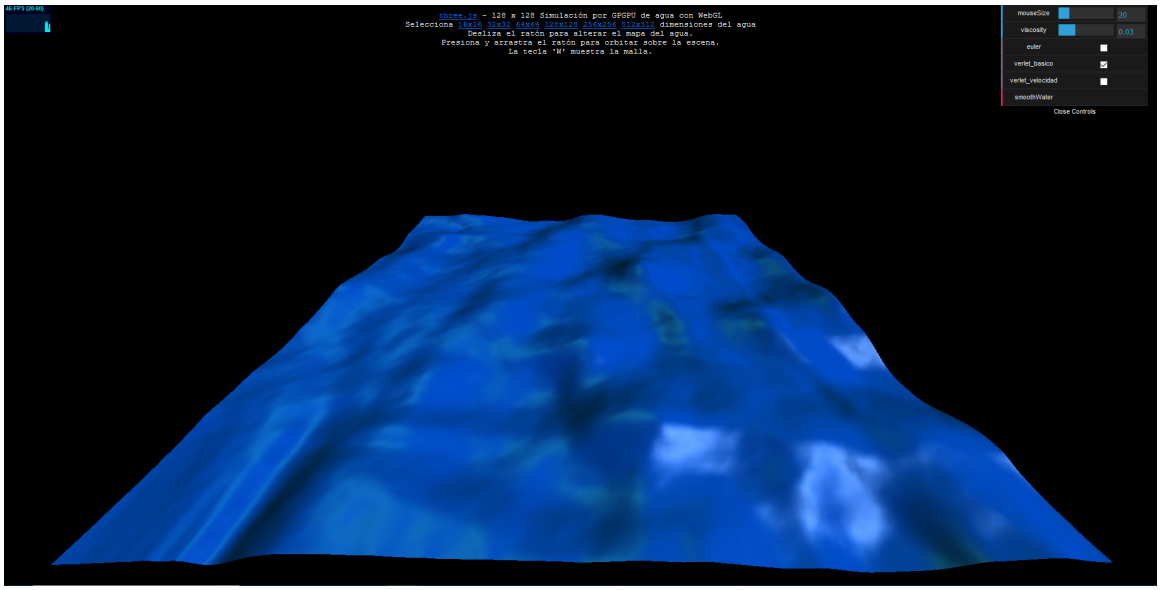


**Figura 6.2:** Captura de la ejecución aplicando el método de Verlet básico o de posición.

### 6.2.3. Método de Verlet con velocidad

El método de Verlet con velocidad integra, implícitamente, el factor de velocidad y añade mejoras que aumentan la precisión y la estabilidad con respecto al método básico de Verlet. Debido a que es una mejora, en principio debe de mantener la estabilidad y convergencia. Sin embargo, no es el resultado obtenido tras la implementación. Al inicio de la ejecución, es bastante similar al método de posición de Verlet. Tras el paso de unos segundos se puede apreciar cómo la escena se desestabiliza ligeramente para que, transcurridos 10-15 segundos, se aprecie con claridad que el método diverge y es inestable. Tras la revisión del código no se ha llegado a encontrar donde se puede estar cometiendo el error. Entre otros, el motivo puede ser debido a no tener en cuenta diversos factores.

Si añadimos ciertas restricciones como pueden ser el de la variación de altura máxima en un paso temporal o la velocidad máxima de variación por intervalo de tiempo, en función a la elección que realicemos, es posible estabilizar la simulación. Con estabilizar hacemos referencia a evitar que la escena sobrepase ciertos valores y tener controlado el rango de movimiento de la malla.

Como se puede ver en la siguiente figura, los nodos de la malla quedan oscilando pero nunca llega a desestabilizarse como en el caso anterior.

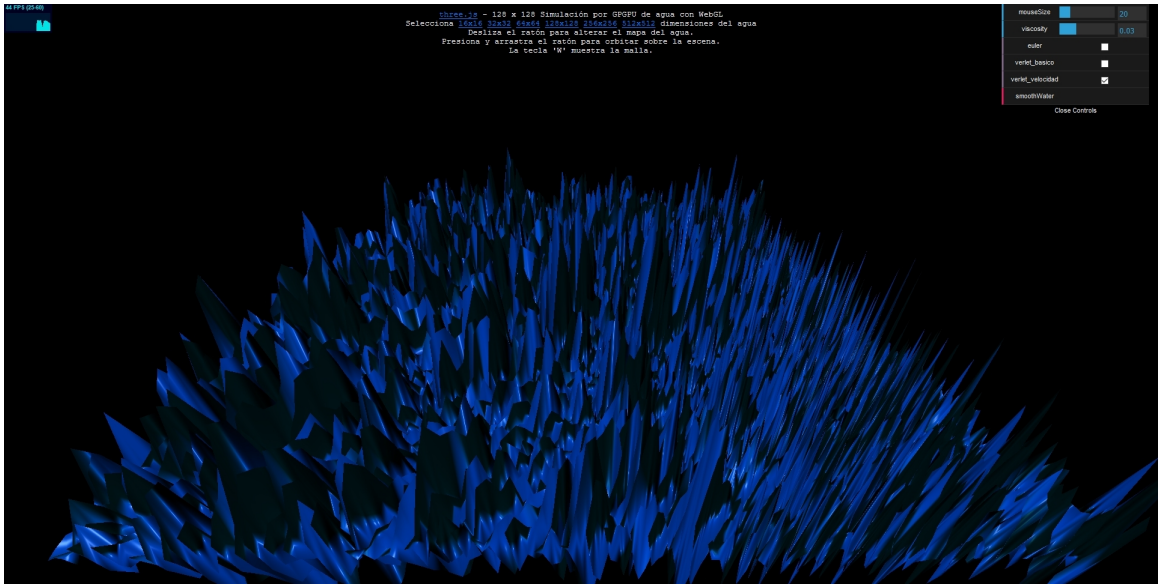


**Figura 6.3:** Captura de la ejecución aplicando el método de Verlet con velocidad (momento de inestabilidad).

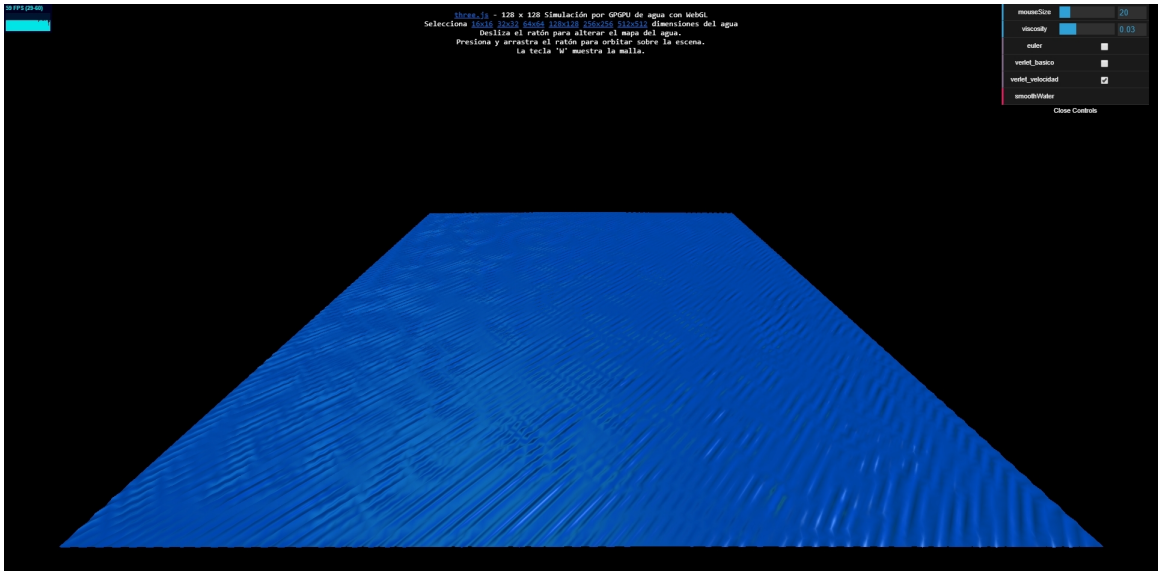


**Figura 6.4:** Captura de la ejecución usando el método de Verlet con velocidad y restricciones.

## 6.3. Interfaz

La interfaz de la simulación consta de distintos elementos. De algunos se ha hablado en puntos anteriores como son el menú de cambio de variables o el contador de FPS, memoria utilizada y tiempo de procesado de un fotograma.

También existe la opción de modificar la escena mediante el ratón. Gracias a este, podremos crear alturas en la malla que tendrán como consecuencia ondas circulares en la simulación. Otra de las posibilidades es la de rotar la escena arrastrando el ratón por la pantalla manteniendo pulsado el clic izquierdo.

Además, existe la opción de modificar el tamaño de la malla o visualizar únicamente la malla en vez de la superficie pulsando la tecla `W`.

# 7. CAPÍTULO

# Conclusiones

## 7.1. Conclusiones generales del proyecto

Tras el desarrollo del proyecto, podemos decir que se han cumplido los objetivos planteados al comienzo del mismo: el estudio de las ecuaciones que describen el comportamiento de los fluidos así como el origen, alcance y profundización de cada una de sus variables, el estudio de los distintos métodos de integración y la comparación de los mismos y la posterior implementación de un simulador de fluidos 3D. Además, hemos trabajado en el desarrollo de programas mediante el uso de la biblioteca «Three.js» escrita en JavaScript y de *shaders* que utilizan texturas como herramienta de cómputo mediante algoritmos de computación GPGPU.

Tras el análisis de los resultados obtenidos una vez realizada la implementación, hemos llegado a la conclusión de que la simulación de fluidos basándose en la física exclusivamente, pese a añadir restricciones, es una tarea más compleja de lo que pueda parecer a priori y existen muchos métodos válidos de llegar a una misma solución como se ha podido ver en este proyecto.

Por otra parte, la simplificación de algunos factores, como la viscosidad, parece no tener tanta influencia en nuestra simulación debido a la configuración de fuerzas que formulan la dinámica de un fluido (en nuestro caso, gravedad y viscosidad) y, por tanto, para que visualmente este factor sea más notable debemos utilizar algún peso para aumentar el efecto de la viscosidad del fluido.

Además, se ha podido ver a través de las implementaciones cómo algunos métodos de integración dependen de distintas condiciones o restricciones para garantizar su estabilidad y que el error que se comete, por pequeño que pueda ser, transcurridos un número elevado de ejecuciones, puede verse magnificado.

## 7.2. Líneas futuras

Debido a la limitación del tiempo a dedicar al proyecto, el escaso conocimiento en el área de la física de fluidos y la necesidad de aprender el correcto funcionamiento de las herramientas que se han utilizado para el desarrollo del proyecto, existen muchos elementos a tener en cuenta para continuar con el estudio en un futuro.

Relacionados con lo desarrollado en este proyecto, los elementos a seguir investigando son la suma de más fuerzas que influyen en el movimiento de fluido y cómo actúan sobre estos, revisar si las restricciones utilizadas para el control de los métodos son los adecuados o existen mejores valores.

También es interesante el estudio de nuevos métodos de integración de mayor precisión que no requieran el uso de condiciones para mantener su estabilidad y que el error cometido en cada paso sea menor o igual a los estudiados.

La implementación realizada ha sido gracias a un mapa de alturas variable y representado por una malla. La simulación de fluidos también se realiza mediante el uso de partículas y es una implementación que se utiliza mucho y cada vez cuenta con mayor presencia en las simulaciones.

# Bibliografía

[Adams et al., 2007] Adams, B., Pauly, M., Keiser, R., and Guibas, L. J. (2007). Adaptively sampled particle fluids. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA. ACM.

[Baboud and Décoret, 2006] Baboud, L. and Décoret, X. (2006). Realistic water volumes in real-time. In *Proceedings of the Second Eurographics Conference on Natural Phenomena*, NPH'06, pages 25–32, Goslar Germany, Germany. Eurographics Association.

[Bargteil et al., 2006] Bargteil, A. W., Goktekin, T. G., O'Brien, J. F., and Strain, J. A. (2006). A semi-lagrangian contouring method for fluid simulation. *ACM Trans. Graph.*, 25(1):19–38.

[Batty et al., 2007] Batty, C., Bertails, F., and Bridson, R. (2007). A fast variational framework for accurate solid-fluid coupling. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA. ACM.

[Batty et al., 2010] Batty, C., Xenos, S., and Houston, B. (2010). Tetrahedral embedded boundary methods for accurate and flexible adaptive fluids. *Computer Graphics Forum*, 29(2):695–704.

[Bridson, 2008] Bridson, R. (2008). *Fluid Simulation for Computer Graphics*. CRC Press.

[Bridson and Müller-Fischer, 2007] Bridson, R. and Müller-Fischer, M. (2007). Fluid simulation: Siggraph 2007 course notesvideo files associated with this course are available from the citation page. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, pages 1–81, New York, NY, USA. ACM.

[Brochu et al., 2010] Brochu, T., Batty, C., and Bridson, R. (2010). Matching fluid simulation elements to surface geometry and topology. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 47:1–47:9, New York, NY, USA. ACM.

[Brochu and Bridson, 2009] Brochu, T. and Bridson, R. (2009). Robust topological operations for dynamic explicit surfaces. *SIAM J. Sci. Comput.*, 31(4):2472–2493.

[Brodtkorb et al., 2010] Brodtkorb, A. R., Sætra, M. L., and Altinakar, M. S. (2010). Efficient shallow water simulations on gpus : Implementation , visualization , verification , and validation.

[Carlson et al., 2004] Carlson, M., Mucha, P. J., and Turk, G. (2004). Rigid fluid: Animating the interplay between rigid bodies and fluid. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 377–384, New York, NY, USA. ACM.

[Chen and da Vitoria Lobo, 1995] Chen, J. X. and da Vitoria Lobo, N. (1995). Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graphical Models and Image Processing*, 57(2):107 – 116.

[Chentanez et al., 2006] Chentanez, N., Goktekin, T. G., Feldman, B. E., and O'Brien, J. F. (2006). Simultaneous coupling of fluids and deformable bodies. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '06, pages 83–89, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Crane et al., 2007] Crane, K., Llamas, I., and Tariq, S. (2007). *Real Time Simulation and Rendering of 3D Fluids*, chapter 30. Addison-Wesley.

[Enright et al., 2005] Enright, D., Losasso, F., and Fedkiw, R. (2005). A fast and accurate semi-lagrangian particle level set method. *Comput. Struct.*, 83(6-7):479–490.

[Enright et al., 2002] Enright, D., Marschner, S., and Fedkiw, R. (2002). Animation and rendering of complex water surfaces. *ACM Trans. Graph.*, 21(3):736–744.

[Enright et al., 2003] Enright, D., Nguyen, D., Gibou, F., and Fedkiw, R. (2003). Using the particle level set method and a second order accurate pressure boundary condition for free surface flows. In *In Proc. 4th ASME-JSME Joint Fluids Eng. Conf., number FEDSM2003-45144. ASME*, pages 2003–45144.

[Fedkiw et al., 2001] Fedkiw, R., Stam, J., and Jensen, H. W. (2001). Visual simulation of smoke. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 15–22, New York, NY, USA. ACM.

[Feldman et al., 2005] Feldman, B. E., O'Brien, J. F., Klingner, B. M., and Goktekin, T. G. (2005). Fluids in deforming meshes. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2005*.

[Foster and Fedkiw, 2001] Foster, N. and Fedkiw, R. (2001). Practical animation of liquids. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 23–30, New York, NY, USA. ACM.

[Foster and Metaxas, 1996] Foster, N. and Metaxas, D. (1996). Realistic animation of liquids. *Graph. Models Image Process.*, 58(5):471–483.

[Fournier and Reeves, 1986] Fournier, A. and Reeves, W. T. (1986). A simple model of ocean waves. *SIGGRAPH Comput. Graph.*, 20(4):75–84.

[Fujisawa et al., 2017] Fujisawa, M., Nakada, T., and Mikawa, M. (2017). Particle-based shallow water simulation with splashes and breaking waves. *Journal of Information Processing*, 25:486–493.

[Gamito and Musgrave, 2002] Gamito, M. N. and Musgrave, F. K. (2002). An accurate model of wave refraction over shallow water. *Computers & Graphics*, 26:291–307.

[Gomez, 2000] Gomez, M. (2000). Interactive simulation of water surfaces. In DeLoura, M., editor, *Game Programming Gems*, pages 187–194. Charles River Media.

[Gourlay, 2012] Gourlay, M. J. (2012). *Fluid Simulation for Video Games (part 1)*.

[Guendelman et al., 2005] Guendelman, E., Selle, A., Losasso, F., and Fedkiw, R. (2005). Coupling water and smoke to thin deformable and rigid shells. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 973–981, New York, NY, USA. ACM.

[Harlow and Welch, 1965] Harlow, F. H. and Welch, J. E. (1965). Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189.

[Hinsinger et al., 2002] Hinsinger, D., Neyret, F., and Cani, M.-P. (2002). Interactive animation of ocean waves. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, pages 161–166, New York, NY, USA. ACM.

[Iglesias, 2004] Iglesias, A. (2004). Computer graphics for water modeling and rendering: A survey. *Future Gener. Comput. Syst.*, 20(8):1355–1374.

[Irving et al., 2006] Irving, G., Guendelman, E., Losasso, F., and Fedkiw, R. (2006). Efficient simulation of large bodies of water by coupling two and three dimensional techniques. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 805–811, New York, NY, USA. ACM.

[Kass and Miller, 1990] Kass, M. and Miller, G. (1990). Rapid, stable fluid dynamics for computer graphics. *SIGGRAPH Comput. Graph.*, 24(4):49–57.

[Klingner et al., 2006] Klingner, B. M., Feldman, B. E., Chentanez, N., and O'Brien, J. F. (2006). Fluid animation with dynamic meshes. In *Proceedings of ACM SIGGRAPH 2006*, pages 820–825.

[Kontaxis, 2013] Kontaxis, C. (2013). Fluid simulation for computer graphics. Master's thesis, Department of Information and Computing Sciences, Utrecht University.

[Lee and Han, 2010] Lee, H. and Han, S. (2010). Solving the shallow water equations using 2d sph particles for interactive applications. 26:865–872.

[Lentine et al., 2010] Lentine, M., Zheng, W., and Fedkiw, R. (2010). A novel algorithm for incompressible flow using only a coarse grid projection. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 114:1–114:9, New York, NY, USA. ACM.

[Losasso et al., 2004] Losasso, F., Gibou, F., and Fedkiw, R. (2004). Simulating water and smoke with an octree data structure. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 457–462, New York, NY, USA. ACM.

[Mastin et al., 1987] Mastin, G. A., Watterberg, P. A., and Mareda, J. F. (1987). Fourier synthesis of ocean scenes. *IEEE Computer Graphics and Applications*, 7(3):16–23.

[McAdams et al., 2010] McAdams, A., Sifakis, E., and Teran, J. (2010). A parallel multigrid poisson solver for fluids simulation on large grids. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 65–74, Goslar Germany, Germany. Eurographics Association.

[Miller and Pearce, 1989] Miller, G. and Pearce, A. (1989). Globular dynamics: A connected particle system for animating viscous fluids. *Computers & Graphics*, 13(3):305 – 309.

[Molemaker et al., 2008] Molemaker, J., Cohen, J. M., Patel, S., and Noh, J. (2008). Low viscosity flow simulations for animation. In *Proceedings of the 2008 ACM SIGGRAP-*

*H/Eurographics Symposium on Computer Animation*, SCA '08, pages 9–18, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Müller, 2009] Müller, M. (2009). Fast and robust tracking of fluid surfaces. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 237–245, New York, NY, USA. ACM.

[Müller et al., 2003] Müller, M., Charypar, D., and Gross, M. (2003). Particle-based fluid simulation for interactive applications. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 154–159, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[O'Brien and Hodgins, 1995] O'Brien, J. F. and Hodgins, J. K. (1995). Dynamic simulation of splashing fluids. In *Proceedings of the Computer Animation*, CA '95, pages 198–, Washington, DC, USA. IEEE Computer Society.

[Osher and Sethian, 1988] Osher, S. and Sethian, J. A. (1988). Fronts propagating with curvature-dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79(1):12 – 49.

[Ottosson, 2011] Ottosson, B. (2011). Real-time interactive water waves. Master's thesis, KTH Computer Science and Communication, Royal Institute of Technology.

[Peachey, 1986] Peachey, D. R. (1986). Modeling waves and surf. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, pages 65–74, New York, NY, USA. ACM.

[PremÅŸoe and Ashikhmin, 2000] PremÅŸoe, S. and Ashikhmin, M. (2000). Rendering natural waters. In *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*, pages 23–434.

[PremÅŸoe et al., 2003] PremÅŸoe, S., Tasdizen, T., Bigler, J., Lefohn, A., and Whitaker, R. T. (2003). Particle-based simulation of fluids. *Computer Graphics Forum*, 22(3):401–410.

[Robinson-Mosher et al., 2008] Robinson-Mosher, A., Shinar, T., Gretarsson, J., Su, J., and Fedkiw, R. (2008). Two-way coupling of fluids to rigid and deformable solids and shells. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 46:1–46:9, New York, NY, USA. ACM.

[Selle et al., 2008] Selle, A., Fedkiw, R., Kim, B., Liu, Y., and Rossignac, J. (2008). An unconditionally stable maccormack method. *Journal of Scientific Computing*, 35(2):350–371.

[Selle et al., 2005] Selle, A., Rasmussen, N., and Fedkiw, R. (2005). A vortex particle method for smoke, water and explosions. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 910–914, New York, NY, USA. ACM.

[Sin et al., 2009] Sin, F., Bargteil, A. W., and Hodgins, J. K. (2009). A point-based method for animating incompressible flow. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 247–255, New York, NY, USA. ACM.

[Solenthaler and Pajarola, 2009] Solenthaler, B. and Pajarola, R. (2009). Predictive-corrective incompressible SPH. In *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, pages 40:1–40:6, New York, NY, USA. ACM.

[Stam, 1999] Stam, J. (1999). Stable fluids. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 121–128, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Stam, 2003] Stam, J. (2003). Real-Time fluid dynamics for games. In *Proceedings of the Game Developer Conference*.

[Stomakhin and Selle, 2017] Stomakhin, A. and Selle, A. (2017). Fluxed animated boundary method. *ACM Trans. Graph.*, 36(4):68:1–68:8.

[Stora et al., 1999] Stora, D., Agliati, P.-O., Cani, M.-P., Neyret, F., and Gascuel, J.-D. (1999). Animating lava flows. In *Graphics Interface (GI'99) Proceedings*, pages 203–210.

[Takahashi et al., 2002] Takahashi, T., Ueki, H., Kunimatsu, A., and Fujii, H. (2002). The simulation of fluid-rigid body interaction. In *ACM SIGGRAPH 2002 Conference Abstracts and Applications*, SIGGRAPH '02, pages 266–266, New York, NY, USA. ACM.

[Tan and Yang, 2009] Tan, J. and Yang, X. (2009). Physically-based fluid animation: A survey. *Science in China Series F: Information Sciences*, 52(5):723–740.

[Terzopoulos et al., 1991] Terzopoulos, D., Platt, J., and Fleischer, K. (1991). Heating and melting deformable models. *The Journal of Visualization and Computer Animation*, 2(2):68–73.

[Tessendorf, 1999] Tessendorf, J. (1999). Simulating ocean water. Technical report, Siggraph Course Notes, ACM Press.

[Thurey et al., 2007] Thurey, N., Müller-Fischer, M., Schirm, S., and Gross, M. (2007). Real-time breakingwaves for shallow water simulations. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications*, PG '07, pages 39–46, Washington, DC, USA. IEEE Computer Society.

[Thürey and Rüde, 2004] Thürey, N. and Rüde, U. (2004). Free surface lattice-boltzmann fluid simulations with and without level sets. In *Proceedings of Vision, Modelling, and Visualization, VMV*.

[Tompson et al., 2016] Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. (2016). Accelerating eulerian fluid simulation with convolutional networks. *CoRR*, abs/1607.03597.

[Ts'o and Barsky, 1987] Ts'o, P. Y. and Barsky, B. A. (1987). Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Trans. Graph.*, 6(3):191–214.

[Valkov et al., 2015] Valkov, B., H. Rycroft, C., and Kamrin, K. (2015). Eulerian method for multiphase interactions of soft solid bodies in fluids. 82.

[Witting, 1999] Witting, P. (1999). Computational fluid dynamics in a traditional animation environment. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 129–136, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

[Wojtan et al., 2010] Wojtan, C., Thürey, N., Gross, M., and Turk, G. (2010). Physics-inspired topology changes for thin fluid features. In *ACM SIGGRAPH 2010 Papers*, SIGGRAPH '10, pages 50:1–50:8, New York, NY, USA. ACM.

[Yngve et al., 2000] Yngve, G. D., O'Brien, J. F., and Hodgins, J. K. (2000). Animating explosions. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 29–36, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.

# A. ANEXO

## Breve introducción a la animación por computador

La animación por computador es la técnica que consiste en crear imágenes en movimiento mediante el uso de una computadora. La animación en 3D frente a la 2D empieza a acaparar el interés total de la demanda en la industria como se puede observar en películas y videojuegos entre otros.

Pionero en este campo fue el animador estadounidense John Witney que, en los años 40 y 50, junto a su hermano James, crearon un conjunto de películas experimentales. Por esta época, únicamente se era capaz de obtener información de imágenes como la extracción de dibujos lineales, contar objetos, reconocer letras o proyectar imágenes.

En 1960, William Fetter, trabajaba en el desarrollo de descripciones ergonómicas del cuerpo humano que fuesen tanto precisos como adaptables a varios ambientes, lo cual dio como resultado la primera figura "wire-frame" tridimensional animada. Estas figuras humanas fueron icono de las primeras gráficas computacionales.

En 1963 se puede decir que se creó una de las primeras animaciones por computador. Creada por Edward Zajac, A Two Gyro Gravity Gradient Attitude Control System mostraba un satélite que podía estabilizarse para que uno de sus lados siempre estuviera en dirección a la tierra mientras éste orbitaba.

Michael Noll creó una película 3D con la técnica de proyección estereográfica en 1965 la cual mostraba un ballet con varios personajes a base de líneas moviéndose por un escenario.

Sin embargo, no fue hasta principios de los años 70 cuando en la facultad de ciencias de Utah se asentaron las técnicas básicas del 3D entre las que caben destacar los sombreados

Gouraud, Phong y Blinn, mapas de texturas, algoritmos de determinación de cara oculta y primeros intentos de realidad virtual entre otros.

En 1968 un grupo de físicos y matemáticos soviéticos crearon un modelo matemático para determinar el movimiento de un gato. Finalmente idearon un programa capaz de resolver ecuaciones diferenciales ordinarias para dicho modelo. De esta forma, se creó el primer personaje animado que consistía en un gato andando.

Scanimate fue la primera máquina que alcanzó fama en los medios. Una computadora diseñada y construida por Lee Harrison que, a partir de 1969, fueron utilizados para producir varias de las animaciones vistas en comerciales y títulos de series de la época. Era capaz de crear animaciones en tiempo real lo que significó un gran avance en esos momentos.

En 1973 se creó el primer largometraje que utilizaba el procesamiento de imágenes digitales. La película de ciencia ficción "Westworld" en la cual varios robots humanoides conviven con humanos para proveerlos de entretenimiento. Se procesaron digitalmente varios fragmentos con el fin de darle un efecto pixelado a algunas escenas.

La década de los 70 fue destacada por los inicios de la animación 3D en cine. La película "Futureworld" mostraba una mano y rostro generados por una computadora. Otra película destacada fue "Star Wars" en 1977, que hizo uso de esta tecnología en las tomas de la Estrella de la Muerte, la mira de las computadoras de las naves X-wing y el famoso Halcón Milenario.

En esta época también se logró un paso más hacia el realismo en la animación 3D gracias al desarrollo de fractales. En 1979 y 1980 se creó la primera película en hacer uso de fractales. "Vol Libre" mostraba un vuelo sobre un paisaje conformado por un fractal.

La animación siguió avanzando hasta la actualidad con nuevas creaciones que, gracias a ellas, actualmente podemos ver en películas con software dedicado para la animación de elementos concretos (por ejemplo en la película "Frozen" en 2013, se creó un software para el diseño de la nieve en todas sus formas), videojuegos en los que se intenta replicar el mundo real de una manera relativamente creíble y la interacción en tiempo real con este.

# B. ANEXO

## Código del proyecto

El código del proyecto está accesible en el repositorio de GitHub en el siguiente enlace: https://github.com/Rive4/WebGL_GPGPU-Water-Simulation