UNIVERSIDADE

DE VIGO

ESCOLA SUPERIOR DE ENXEÑARÍA INFORMÁTICA

Memoria do Traballo de Fin de Máster que presenta

D. Iván Martínez Estévez

para a obtención do Título de Máster en Enxeñaría Informática

Acoplamiento del modelo DualSPHysics con una librería multifísica

Xaneiro, 2021

Traballo de Fin de Máster Nº: MEI 20/21-6

Titor/a: Javier Rodeiro Iglesias
Cotitor/a: José Manuel Domínguez Alonso
Área de coñecemento: Linguaxes e Sistemas Informáticos
Departamento: Informática

Agradecimientos

Me gustaría dedicarle unas palabras a todas las personas que me han ayudado.

A Jose y Álex, por seguir confiando en mí. Por vuestros consejos a lo largo de todo este tiempo y vuestra dedicación, por guiarme y ayudarme en todo momento siempre que fue necesario.

A mi tutor Javier, por haber aceptado dirigirme este trabajo y haber confiado en mí una vez más.

A mis compañeros del grupo EPhysLab, por hacerme sentir como uno más desde el momento en el que incorporé.

A Irene, mi pareja, por estar siempre a mi lado, apoyarme y aconsejarme en todas las decisiones importantes.

A mi familia, tanto a los a los que están presentes como los que han estado a lo largo de mi vida, con especial mención a mi hermana. Gracias por estar siempre a mi lado, confiar en mí y ayudarme en todos los aspectos de la vida.

A todos vosotros, gracias.

Iván Martínez Estévez

ÍNDICE DE CONTENIDOS

ÍNDICE DE CONTENIDOS i
ÍNDICE DE FIGURAS iii
ÍNDICE DE TABLAS vi
LISTA DE ACRÓNIMOS vii
INTRODUCCIÓN 1
OBJETIVOS 2
DESCRIPCIÓN DEL TRABAJO 3
1 METODOLOGÍA 3
1.1 SOLUCIÓN TÉCNICA 4
1.2 PLAN DE TRABAJO 5
2 PLANIFICACIÓN Y SEGUIMIENTO 7
2.1 PLANIFICACIÓN 7
2.2 PUNTOS CRÍTICOS 8
2.3 EJECUCIÓN 8
2.4 DESVIACIONES 9
3 ARQUITECTURA 10
4 TECNOLOGÍAS Y HERRAMIENTAS 11
5 ESPECIFICACIÓN Y ANÁLISIS DE REQUISITOS 14
5.1 ITERACIÓN 0. ESTUDIO DEL PROBLEMA 14
5.2 ITERACIÓN 3. ACOPLAMIENTO CON DSPHCHRONOLIB 15
5.3 ITERACIÓN 4. ACOPLAMIENTO CON CHRONO 16
5.4 ITERACIÓN 5. SALIDA DE DATOS 17
5.5 ITERACIÓN 6. MÓDULO PARALELO 18
5.6 ITERACIÓN 7. COLISIONES 19
5.7 ITERACIÓN 8. ENLACES ENTRE OBJETOS 20
6 DISEÑO DEL SOFTWARE 22
6.1 DISEÑO ESTÁTICO 23
6.2 DISEÑO DINÁMICO 24
7 GESTIÓN DE DATOS E INFORMACIÓN 30
8 PRUEBAS LLEVADAS A CABO 32
8.1 ITERACIÓN 3. ACOPLAMIENTO CON DSPHCHRONOLIB 32
8.2 ITERACIÓN 4. ACOPLAMIENTO CON CHRONO 33
8.3 ITERACIÓN 5. SALIDA DE DATOS 34
8.4 ITERACIÓN 6. MÓDULO PARALELO 35
8.5 ITERACIÓN 7. COLISIONES 37
8.6 ITERACIÓN 8. ENLACES ENTRE OBJETOS 39

8.7 PRUEBAS GLOBALES 41
9 MANUAL DE USUARIO 43
9.1 COMPILACIÓN 43
9.2 EJECUCIÓN 47
CONCLUSIONES 49
REFERENCIAS 50
ANEXO 1. FLUJO DE TRABAJO DE DUALSPHYSICS 1
ANEXO 2. ESTRUCTURA DE DUALSPHYSICSCHRONO 2
ANEXO 3. COMPILACIÓN DE CHRONO 3
ANEXO 4. COMPILACIÓN DE DUALSPHYSICS 8

ÍNDICE DE FIGURAS

Figura 1. Tareas de las iteraciones de implementación 3
Figura 2. Esquema de la solución propuesta 4
Figura 3. Diagrama de Gantt de planificación donde se muestran los puntos críticos (rojo) 7
Figura 4. Diagrama de Gantt de seguimiento donde se muestran las desviaciones que suponen reducciones de tiempo (verde) y las que suponen atrasos (rojo) 8
Figura 5. Arquitectura del acoplamiento 10
Figura 6. Opción para habilitar y configurar la exportación de datos en ficheros CSV 17
Figura 7. Selección del modo de ejecución en Chrono 18
Figura 8. Selección del método de contacto 19
Figura 9. Uso de la opción de imponer coeficientes de fricción 19
Figura 10. Definición de un link Coulomb damping 20
Figura 11. Definición de un link Pulley 20
Figura 12. Coeficientes variables de damping y stiffness desde CSV 21
Figura 13. Coeficientes variables de stiffness y damping embebido en el XML 21
Figura 14. Diagrama de Clases 23
Figura 15. Diagrama de Secuencia del acoplamiento con DSPHChronoLib 24
Figura 16. Diagrama de Secuencia del acoplamiento con Chrono 25
Figura 17. Diagrama de Secuencia de la salida de datos 26
Figura 18. Diagrama de Secuencia del módulo paralelo 27
Figura 19. Diagrama de Secuencia de las colisiones 28
Figura 20. Diagrama de Secuencia de los enlaces entre objetos 29
Figura 21. Plantilla de configuración de Chrono 30
Figura 22. Configuración del caso de validación del acoplamiento con DSPHChronoLib 32
Figura 23. Salida por consola del caso de validación del acoplamiento con DSPHChronoLib 32
Figura 24. Configuración del caso de validación del acoplamiento con Chrono 33
Figura 25. Secuencia de simulación para la validación del acoplamiento con Chrono 33
Figura 26. Configuración del caso de validación de la salida de datos 34
Figura 27. Ficheros CSV generados en el caso de validación de la salida de datos 34
Figura 28. Configuración del caso de validación del módulo paralelo 35
Figura 29. Salida por consola del caso de validación del módulo paralelo 35
Figura 30. Secuencia de simulación del caso de validación del módulo paralelo 36
Figura 31. Configuración del caso de validación de las colisiones SMC 37
Figura 32. Secuencia de simulación para la validación de las colisiones SMC 37
Figura 33. Centro de masas en Z del objeto flotante con dos métodos de colisión 37
Figura 34. Gráfica de la posición en Z del objeto flotante con colisiones NSC 38

Figura 35. Configuración del caso de validación del link Coulomb damping 39
Figura 36. Secuencia de simulación del link Coulomb damping 39
Figura 37. Configuración del caso de validación del link pulley 40
Figura 38. Secuencia de simulación del link pulley 40
Figura 39. Elevación en Z del objeto flotante conectado a un muelle lineal 41
Figura 40. Dimensiones del dominio del caso de validación global 41
Figura 41. Configuración del caso de validación global 41
Figura 42. Secuencia de la simulación del caso de validación global 42
Figura 43. Valores de desplazamiento experimentales (EXP) y numéricos (DualSPHysics) 42
Figura 44. Introducir las rutas para la generación de DSPHChronoLib 44
Figura 45. Selección de compiladores en Windows para DSPHChronoLib 44
Figura 46. Selección de compiladores en Linux para DSPHChronoLib 44
Figura 47. Introducir la ruta de la librería Chrono 45
Figura 48. Introducir la ruta de la librería Blaze 45
Figura 49. Generación de la solución de DSPHChronoLib 46
Figura 50. Selección del modo de compilación de DSPHChronoLib 46
Figura 51.Compilar DSPHChronoLib con Microsoft Visual Studio 2015 46
Figura 52. Compilar DSPHChronoLib con Make en Linux 47
Figura 53. Command prompt de Windows 48
Figura 54. Ejecución de una simulación de DualSPHysics en Windows 48
Figura 55. Abrir un terminal de Linux en DualSPHysicsChrono 48
Figura 56. Dar permisos de ejecución a scripts y ejecutables de Linux 48
Figura 57. Acceder al directorio 00_CaseManual 48
Figura 58. Ejecución de una simulación de DualSPHysics en Linux 48
Figura 59. Flujo de trabajo de DualSPHysics 1
Figura 60. Estructura del directorio DualSPHysicsChrono 2
Figura 61. Introducir las rutas para la generación de Chrono 3
Figura 62. Selección de compiladores en Windows para Chrono 4
Figura 63. Selección de compiladores en Linux para Chrono 4
Figura 64. Seleccionar la ubicación de la configuración de CMake 4
Figura 65. Habilitar el uso del módulo PARALLEL 5
Figura 66. Introducir las rutas de la librería Blaze, Boost y Thrust 5
Figura 67. Generación de la solución de Chrono 6
Figura 68. Selección del modo de compilación de Chrono 6
Figura 69. Compilar Chrono con Microsoft Visual Studio 2015 6
Figura 70. Compilar Chrono con Make en Linux 7
Figura 71. Habilitar/deshabilitar el uso de librerías externas en DualSPHysics 8

Figura 72. Selección del modo de compilación para CPU&GPU en DualSPHysics 8
Figura 73. Selección del modo de compilación para CPU en DualSPHysics 8
Figura 74. Compilar DualSPHysics con Microsoft Visual Studio 2015 9
Figura 75. Compilar DualSPHysics para CPU&GPU con Make en Linux 9
Figura 76. Compilar DualSPHysics para CPU con Make en Linux..... 9

ÍNDICE DE TABLAS

Tabla 1. Plan de trabajo 5
Tabla 2. Planificación del trabajo 7
Tabla 3. Tiempos de ejecución y planificación del trabajo 8
Tabla 4. Estructura del fichero CSV de datos intercambiados entre DualSPHysics y Chrono ... 30
Tabla 5. Estructura del fichero CSV de fuerzas 31
Tabla 6. Tiempos de simulación del caso de validación del módulo paralelo 36

LISTA DE ACRÓNIMOS

CFD. Dinámica de fluidos computacional (*Computational Fluid Dynamics*).

CPU. Unidad central de procesamiento (*Central Processing Unit*).

CSV. Comma-separated values.

GCC. GNU Compiler Collection.

GDB. GNU Debugger.

GPU. Unidad de Procesamiento Gráfico (*Graphics Processing Unit*).

IDE. Entorno de desarrollo integrado (*Integrated Development Environment*).

MVS. Microsoft Visual Studio.

NSC. Non-Smooth Contacts. Contactos basados en restricciones rígidas (*constraint-based*) o de cuerpo rígido (*rigid-body*).

SIMD. Una instrucción, múltiples datos (*Single Instruction, Multiple Data*).

SMC. SMooth Contacts. Contactos basados en penalización (*penalty-based*) o de cuerpo blando (*soft-body*).

SPH. Smoothed Particle Hydrodynamics.

UML. Lenguaje unificado de modelado (*Unified Modelling Language*).

VP. Visual Paradigm.

VTK. The Visualization Toolkit.

XML. eXtensible Markup Language.

MEMORIA

INTRODUCCIÓN

Este trabajo de Fin de Máster (TFM) ha sido realizado en el grupo de investigación EPhysLab (*Environmental Physics Laboratory*), del Campus de Ourense, de la Universidad de Vigo.

Actualmente la dinámica de fluidos computacional (CFD por sus siglas en inglés) es una herramienta informática fundamental en multitud de campos. Permite realizar ensayos numéricos sin la necesidad de construir modelos a escala o prototipos reales con el consiguiente ahorro en tiempo y costes económicos. Los modelos numéricos no pueden sustituir a los ensayos físicos, pero permiten reducir significativamente su número y al mismo tiempo pueden proporcionar información difícil o imposible de medir en ensayos reales.

Los modelos CFD se pueden dividir en modelos basados en malla y en modelos libres de malla. Existen distintos modelos numéricos de malla muy conocidos, robustos y ampliamente utilizados como los basados en Volúmenes Finitos o Elementos Finitos. Sin embargo, en los últimos años los modelos sin malla han ganado una gran popularidad porque pueden resolver algunos problemas donde los modelos tradicionales de malla fallan o no son eficientes. Entre los modelos sin malla, cabe destacar el método *Smoothed Particle Hydrodynamics* (SPH). SPH es un método de partículas donde los cálculos se llevan a cabo en nodos (partículas) que pueden desplazarse libremente por el sistema. Esta característica permite al método SPH realizar simulaciones de interacciones entre fluido y estructuras con gran precisión, donde la superficie libre no requiere un tratamiento especial y se pueden resolver altas deformaciones de ésta sin los problemas que aparecerían en los modelos de malla cuando grandes deformaciones de la malla pueden dar lugar a inconsistencias y errores de ejecución.

El modelo DualSPHysics [1] (https://dual.sphysics.org) es un CFD basado en el método SPH, orientado a la simulación de fluidos con superficie libre y su interacción con estructuras fijas y flotantes. Está desarrollado por el grupo EPhysLab, en colaboración con The University of Manchester, Instituto Superior Tecnico de Lisboa, Università degli studi di Parma, Flanders Hydraulics Research, Universitat Politécnica de Catalunya y New Jersey Institute of Technology. Es un modelo de código abierto con licencia LGPL que cuenta con un importante reconocimiento por parte de la comunidad en el campo de la ingeniería costera. Es un referente en su campo y ya fue aplicado con éxito en múltiples proyectos de investigación y empresariales [2]–[4]. DualSPHysics está basado en un modelo anterior, denominado SPHysics [5], [6] que estaba implementado en Fortran. SPHysics ofrecía resultados muy precisos, pero era muy lento para simular casos con un gran número de partículas debido al coste computacional que tiene asociado el método SPH. DualSPHysics es una implementación paralelizada de SPH en C++ y CUDA [7] que permite ejecutar este modelo sobre la CPU o varias veces más rápido en GPU. En el ANEXO 1. FLUJO DE TRABAJO DE DUALSPHYSICS se puede consultar cómo es la secuencia de simulación de este modelo y las herramientas utilizadas en el proceso de ejecución.

Chrono [8] (https://projectchrono.org) es una librería de código abierto que proporciona un motor de simulación multifísica. Las áreas de aplicación en las que Chrono se utiliza con mayor frecuencia son la dinámica de vehículos, la robótica y el diseño de máquinas. Esta librería es capaz de simular cientos de problemas mecánicos de diferente complejidad con gran precisión y eficiencia, tales como: objetos rígidos y deformables, detección de colisiones, soporte de fricción, resortes y amortiguadores.

Normalmente, el uso de un único modelo no es suficiente para simular problemas reales complejos porque intervienen distintos procesos o mecanismos físicos no resueltos por un CFD. Por lo tanto, surge la necesidad de acoplar el CFD, en este caso DualSPHysics, con una librería multifísica que permita llevar a cabo simulaciones más complejas. En este trabajo se desarrollará una estrategia general que facilite el acoplamiento con distintos modelos y en especial se dotará a DualSPHysics de algunas de las funcionalidades más avanzadas que posee Chrono, además de mantener la precisión y eficiencia de ambos modelos trabajando conjuntamente.

Antes de la realización de este TFM, ya existía un acoplamiento entre DualSPHysics y Chrono [9]. Este acoplamiento, basado en la versión 2.0 de Chrono del año 2014, permitía la simulación de objetos rígidos y algunos tipos de enlaces o *links* entre objetos como son muelles lineales. Sin embargo, este acoplamiento era muy limitado y sólo incluía las funciones más básicas de Chrono. Además, el código fuente de esta versión no fue actualizado desde entonces y no incluía las nuevas características disponibles en las últimas versiones de Chrono. Por esta razón, es necesario realizar un acoplamiento con una versión más reciente y estable que incorpore nuevas funcionalidades al modelo DualSPHysics. También se realizarán cambios en el código original de Chrono para agregar nuevas funcionalidades necesarias para casos de aplicación frecuentes de DualSPHysics y no contempladas por Chrono. El acoplamiento con Chrono se realizará mediante una librería dinámica que actúe de interfaz de comunicación entre DualSPHysics y Chrono. Además, se seguirá una estrategia que facilite el acoplamiento de DualSPHysics con futuras versiones de Chrono.

OBJETIVOS

El objetivo principal de este trabajo es realizar un nuevo y mayor acoplamiento del modelo DualSPHysics con la librería multifísica Chrono. Este objetivo está dividido en una serie de subobjetivos:

Objetivo 1. Mantener el flujo de comunicación existente en el acoplamiento antiguo entre DualSPHysics y Chrono.

Objetivo 2. Mantener la compatibilidad y funciones ya disponibles en el acoplamiento antiguo.

Objetivo 3. Buscar una estrategia de acoplamiento multiplataforma que facilite el correcto funcionamiento de DualSPHysics, independientemente de la versión de Chrono utilizada.

Objetivo 4. Reducir el tiempo de ejecución de simulaciones complejas donde intervengan un número elevado de objetos colisionando entre sí.

Objetivo 5. Permitir la resolución de las colisiones teniendo en cuenta contactos elásticos (*SMooth Contacts*, SMC) y rígidos (*Non-Smooth Contacts*, NSC).

Objetivo 6. Permitir variar el comportamiento de las colisiones, definiendo el valor del coeficiente de fricción (*friction*) empleado para resolver el rozamiento entre dos objetos.

Objetivo 7. Añadir nuevos tipos de enlaces o *links* entre objetos para dotar al acoplamiento de más características y funcionalidades.

Objetivo 8. Permitir la variación de los coeficientes de amortiguación (*damping*) y elasticidad (*stiffness*) en tiempo de ejecución desde un fichero externo CSV o desde el fichero de configuración de entrada XML.

Objetivo 9. Generar una salida de información con un formato que sea compatible con cualquier herramienta de visualización y análisis de datos.

DESCRIPCIÓN DEL TRABAJO

En este apartado se incluye la documentación técnica relativa al desarrollo software realizado en este trabajo.

1 METODOLOGÍA

Se ha decidido utilizar el modelo Iterativo Incremental [10] como metodología de desarrollo software. Esta metodología permite ir refinando los requisitos a medida que se avanza en el desarrollo del proyecto. Se considera apropiado el uso de este modelo porque no todos los requisitos están definidos completamente en un primer momento y porque permite la entrega de versiones funcionales al cliente a medida que se agregan incrementos.

El ciclo de vida de este modelo se divide en etapas, donde cada etapa se corresponde a un periodo de tiempo del proceso de desarrollo software que da lugar a un conjunto de hitos o entregables. Cada etapa se divide en fases, donde cada una engloba a un conjunto de tareas a realizar para la consecución del proyecto. Al ser un modelo Iterativo e Incremental, el conjunto de tareas a llevar a cabo se agrupará en iteraciones.

El modelo Iterativo Incremental se compone de las siguientes etapas:

- Etapa de inicialización.
- Etapa de iteración.
- Lista de control del proyecto.

En la etapa de inicialización se han definido las fases de análisis y diseño. En la fase de análisis se llevará a cabo el análisis de requisitos inicial. En la fase de diseño se modelará el diseño estático del software y la arquitectura del sistema.

La etapa de iteración está compuesta por las fases de implementación y finalización [10]. La fase de implementación englobará todas las iteraciones que añadan nuevas funcionalidades al sistema. En la fase de finalización se llevarán a cabo las pruebas globales y la documentación final.

En las iteraciones de la fase de implementación se llevarán a cabo las siguientes tareas:

- Análisis de requisitos.
- Diseño.
- Implementación.
- Integración.
- Validación del sistema.

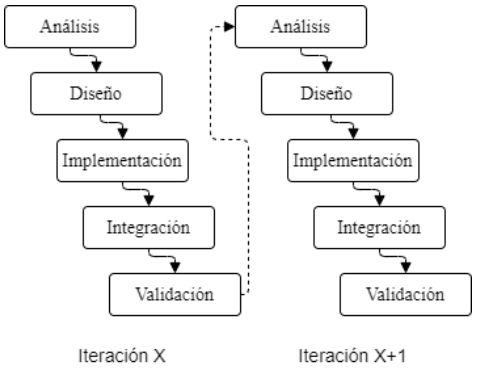

Figura 1. Tareas de las iteraciones de implementación

La lista de control del proyecto servirá de guía para la realización del trabajo. En ella se irán reflejando las tareas finalizadas y las nuevas funcionalidades a implementar que puedan surgir a lo largo del trabajo.

1.1 SOLUCIÓN TÉCNICA

En la Figura 2 se muestra el esquema de las partes que se implementarán en este trabajo (fondo azul). También se realizarán modificaciones en los códigos de DualSPHysics y Chrono para incorporar nuevas funcionalidades.

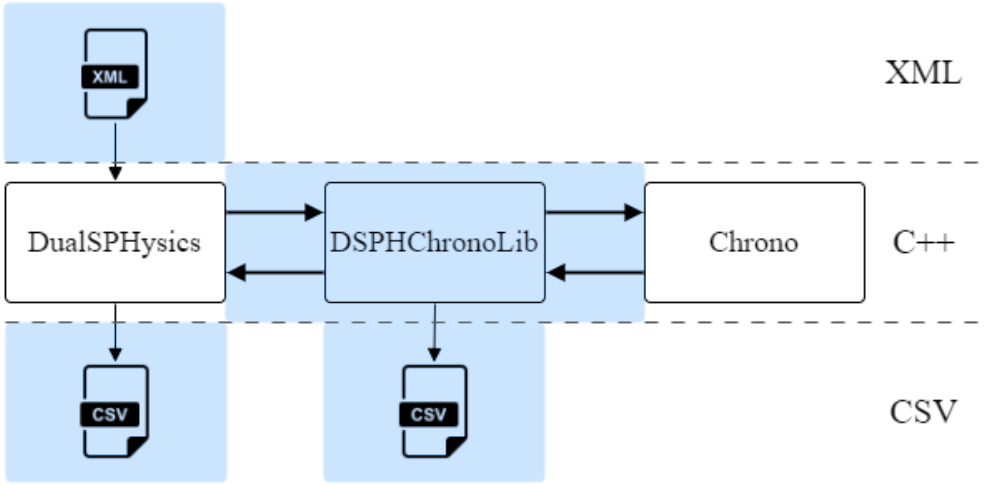

Figura 2. Esquema de la solución propuesta

A lo largo del trabajo se desarrollará una librería dinámica denominada DSPHChronoLib que permitirá el acoplamiento entre DualSPHysics y Chrono, manteniendo el flujo de comunicación existente en la versión antigua. Se utilizará el lenguaje de programación C++ 14 y se compilará con Microsoft Visual C++ 2015 en plataformas Windows, ya que es la última versión utilizada en DualSPHysics y con G++ 10.2.0 en plataformas Linux.

1.2 PLAN DE TRABAJO

En este apartado se describe el trabajo a realizar aplicando la metodología de desarrollo software seleccionada. La Tabla 1 muestra como se ha organizado el plan de trabajo.

Etapa	Fase	Iteración
Inicialización	Análisis	**Iteración 0**. Estudio del problema
	Diseño	**Iteración 1**. Diseño de clases
		Iteración 2. Arquitectura
Iteración	Implementación	**Iteración 3**. Acoplamiento con DSPHChronoLib
		Iteración 4. Acoplamiento con Chrono
		Iteración 5. Salida de datos
		Iteración 6. Módulo paralelo
		Iteración 7. Colisiones
		Iteración 8. Enlaces entre objetos
	Finalización	**Iteración 9**. Pruebas globales
		Iteración 10. Documentación final
Lista control proyecto		

Tabla 1. Plan de trabajo

En la fase de análisis se realizará un estudio del problema (Iteración 0) que dará lugar a los apartados INTRODUCCIÓN, OBJETIVOS, METODOLOGÍA, PLANIFICACIÓN, PUNTOS CRÍTICOS.

En la fase de diseño se modelará el diseño estático del acoplamiento (Iteración 1) y la arquitectura del sistema (Iteración 2). El diseño del software se irá refinando a medida que se agreguen los incrementos de las iteraciones de implementación.

En la fase de implementación se llevarán a cabo todas las iteraciones que agreguen nuevas funcionalidades, es decir, en las que se realizará toda la codificación de este trabajo. Esta fase contiene seis iteraciones (desde la Iteración 3 hasta la Iteración 5). A continuación, se describe el trabajo a realizar en cada una de las iteraciones de esta fase de implementación.

Iteración 3. Acoplamiento con DSPHChronoLib

Se desarrollará la librería dinámica DSPHChronoLib que actuará de interfaz de comunicación entre DualSPHysics y Chrono. Esta librería será multiplataforma y garantizará la adaptabilidad del acoplamiento a sistemas operativos como Windows y Linux. Se implementará un fichero *CMakeLists.txt* para la construcción y generación de la librería usando CMake. El uso de DSPHChronoLib como interfaz de comunicación garantizará que DualSPHysics funcione correctamente independientemente de la versión de Chrono utilizada. En esta iteración, se cumplirá el Objetivo 3.

Iteración 4. Acoplamiento con Chrono

En esta iteración se realizará el acoplamiento entre DualSPHysics y Chrono por medio de la librería DSPHChronoLib, manteniendo el flujo de comunicación existente en el acoplamiento antiguo (Objetivo 1). Se implementará la inicialización, configuración, lectura y la ejecución del entorno de Chrono en DSPHChronoLib. En esta iteración también se incorporarán las funcionalidades existentes en el acoplamiento antiguo, cumpliendo el Objetivo 2.

Iteración 5. Salida de datos

Se implementará la salida de datos del sistema mediante la generación de ficheros en formato CSV. Estos ficheros almacenarán la información intercambiada entre DualSPHysics y Chrono y las fuerzas utilizadas por Chrono para desplazar los objetos. El formato CSV facilita la lectura y análisis de los datos generados y es compatible con cualquier herramienta de visualización, cumpliendo el Objetivo 9.

Iteración 6. Módulo paralelo

El código fuente de Chrono está divido en módulos. Cada uno ofrece una serie de funcionalidades que complementan al código base. Se añadirá el módulo *PARALLEL* que permitirá habilitar la computación paralela con el fin de acelerar los cálculos realizados en Chrono y reducir los tiempos de simulación. Incorporando este módulo, se cumplirá el Objetivo 4.

Iteración 7. Colisiones

Se añadirá la posibilidad de realizar simulaciones con métodos de contacto basados en colisiones elásticas, denominados *SMooth Contacts* (SMC) en Chrono. De esta forma se podrán simular colisiones elásticas (SMC) y rígidas (NSC). Permitiendo el uso de ambos tipos de colisiones, se cumplirá el Objetivo 5.

Se realizarán modificaciones en el código original de Chrono para poder imponer coeficientes de fricción cuando se resuelve el rozamiento entre dos objetos. Estas modificaciones se realizarán para ambos tipos de colisiones (elásticas y rígidas) y para los dos modos de ejecución (secuencial y paralela). Esta característica permitirá variar el comportamiento de las colisiones cuando se produce el rozamiento entre dos objetos, cumpliendo el Objetivo 6.

Iteración 8. Enlaces entre objetos

En esta iteración se incorporarán dos nuevos tipos de enlaces o *links* entre objetos. Uno de ellos estará basado en un muelle lineal (*linear spring*) con modificaciones en su formulación para poder añadir una fuerza de Coulomb (C*oulomb damping*). El otro *link* será una polea (*Pulley*). La incorporación de estos nuevos *links* dotará de nuevas características al acoplamiento y se cumplirá el Objetivo 7.

Se implementará la lectura de valores de coeficientes de amortiguación (*damping*) y elasticidad (*stiffness*) para los *links*, desde un fichero externo CSV o desde el propio fichero de configuración de DualSPHysics. De esta forma, se permitirá la asignación de coeficientes variables durante la ejecución, cumpliendo el Objetivo 8.

En la fase de finalización se llevarán a cabo las pruebas globales del sistema (Iteración 9) que servirán para validar el sistema y la documentación final (Iteración 10), donde se terminarán y perfilarán detalles de la documentación necesaria para la finalización del trabajo.

Se utilizará el lenguaje unificado de modelado (*Unified Modeling Language*, UML) para la representación del sistema, ya que está diseñado para visualizar, especificar, construir y documentar software orientado a objetos [11].

2 PLANIFICACIÓN Y SEGUIMIENTO

En este apartado se documenta la planificación y seguimiento del trabajo aplicando la metodología de software elegida. También se documentan los puntos críticos definidos y las desviaciones entre la planificación y la ejecución del trabajo.

2.1 PLANIFICACIÓN

La Tabla 2 muestra la planificación inicial de la realización del trabajo, donde se puede observar el tiempo estimado de duración de cada iteración. Cabe destacar que se planifica dedicar 4 horas de trabajo diario. Se establece el día 19 de octubre de 2020 como fecha de arranque del proyecto y el día 12 de enero de 2021 como fecha prevista de finalización, tal como se puede observar en la Figura 3.

Etapa	Fase	Iteración	Tiempo estimado (Horas)
Inicialización	Análisis	**Iteración 0**. Estudio del problema	40
	Diseño	**Iteración 1**. Diseño de clases	20
		Iteración 2. Arquitectura	4
Iteración	Implementación	**Iteración 3**. Acoplamiento con DSPHChronoLib	20
		Iteración 4. Acoplamiento con Chrono	40
		Iteración 5. Salida de datos	6
		Iteración 6. Módulo paralelo	32
		Iteración 7. Colisiones	20
		Iteración 8. Enlaces entre objetos	15
	Finalización	**Iteración 9**. Pruebas globales	8
		Iteración 10. Documentación final	12
Lista control proyecto			8

Tabla 2. Planificación del trabajo

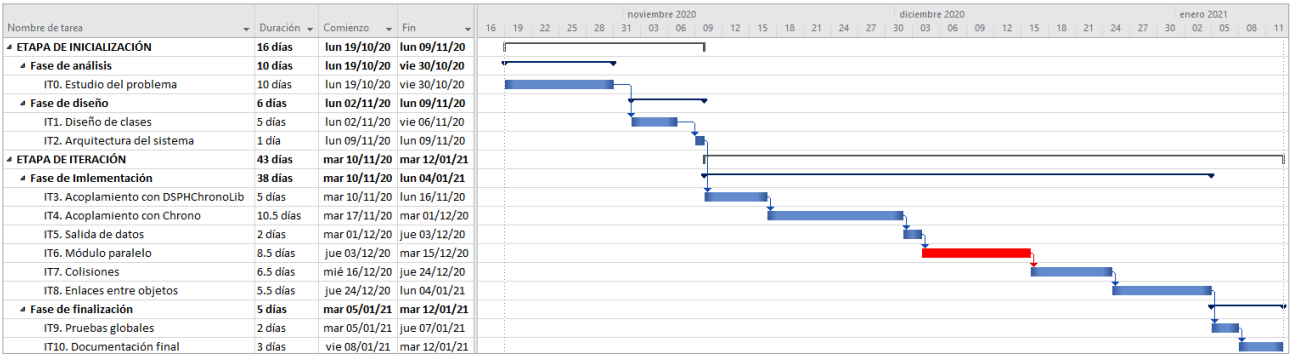

Figura 3. Diagrama de Gantt de planificación donde se muestran los puntos críticos (rojo)

2.2 PUNTOS CRÍTICOS

Se ha definido un punto crítico en la planificación, tal como se puede observar en la Figura 3, en color rojo. Se corresponde con la Iteración 6, donde se agregará el módulo de computación paralela. Este módulo utiliza clases y solucionadores diferentes al código base secuencial y su compilación genera una nueva librería dinámica de Chrono (*ChronoEngine_parallel*). La inclusión de este módulo variará significativamente las funciones principales de comunicación con Chrono, lo que implica la necesidad de separar los accesos a los códigos secuencial y paralelo en DSPHChronoLib. El uso de este módulo modificará el comportamiento de DualSPHysics, ya que será necesario adaptar el acoplamiento para el uso de nuevas librerías dinámicas. También se tendrán que realizar cambios para poder compilar todo el sistema con o sin módulos adicionales de Chrono.

2.3 EJECUCIÓN

La Tabla 3 muestra las horas ejecutadas en cada una de las iteraciones. Existen diferencias con respecto a la planificación inicial provocadas por desviaciones en algunas iteraciones. Las iteraciones que presentan desviaciones se pueden observar en la Figura 4 de color verde y rojo.

Etapa	Fase	Iteración	Tiempo estimado (Horas)	Tiempo ejecución (Horas)
Inicialización	Análisis	**Iteración 0.** Estudio del problema	40	32
	Diseño	**Iteración 1.** Diseño de clases	20	20
		Iteración 2. Arquitectura	4	4
Iteración	Implementación	**Iteración 3.** Acoplamiento con DSPHChronoLib	20	20
		Iteración 4. Acoplamiento con Chrono	40	40
		Iteración 5. Salida de datos	6	4
		Iteración 6. Módulo paralelo	32	40
		Iteración 7. Colisiones	20	20
		Iteración 8. Enlaces entre objetos	15	19
	Finalización	**Iteración 9.** Pruebas globales	8	8
		Iteración 10. Documentación final	12	12
Lista control proyecto			8	8

Tabla 3. Tiempos de ejecución y planificación del trabajo

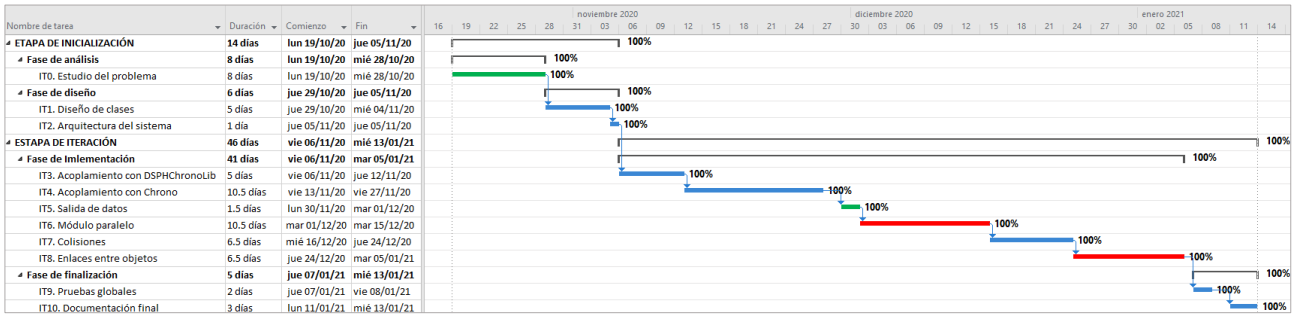

Figura 4. Diagrama de Gantt de seguimiento donde se muestran las desviaciones que suponen reducciones de tiempo (verde) y las que suponen atrasos (rojo)

2.4 DESVIACIONES

A lo largo del trabajo se han producido desviaciones que ocasionaron diferencias entre los tiempos planificados y los tiempos de ejecución, tal como se muestra en la Tabla 3.

En la fase de análisis se han invertido 8 horas menos de lo planificado inicialmente para el estudio del problema.

En la fase de implementación se han producido variaciones en distintas iteraciones que dan lugar a un retraso de 10 horas. En la Iteración 5 se ha implementado la salida de datos mediante generación de los ficheros, invirtiendo 2 horas menos. En la Iteración 6 se han necesitado 8 horas más de las planificadas, debido a la necesidad de realizar cambios en el código original de Chrono. Este módulo se compila por defecto haciendo uso de instrucciones SIMD (*Single Instruction, Multiple Data*) que causaban errores al intentar ejecutarse en equipos con procesadores que no las soportaban. Por ello, fue necesario realizar modificaciones para evitar que se habiliten por defecto. En la Iteración 8 se han invertido 4 horas más al incorporar el *link pulley*, debido a que surgieron problemas en su implementación. Se detectaron y corrigieron defectos de programación en el código original de Chrono que causaban *segmentation fault* o "violación de acceso" al intentar acceder a direcciones de memoria no permitidas.

En definitiva, el trabajo se ha retrasado 2 horas sobre lo planificado debido a las desviaciones en ambas fases, lo que ha provocado que se haya terminado un día más tarde de la fecha marcada como fin de proyecto.

3 ARQUITECTURA

La Figura 5 muestra la arquitectura del sistema. Se definen 3 partes bien diferenciadas, todas ellas desarrolladas en C++: DualSPHysics, DSPHChronoLib y Chrono. DualSPHysics será el software que guiará la ejecución, mientras que DSPHChronoLib y Chrono serán librerías dinámicas.

El sistema se iniciará en el Sistema Operativo (SO) lanzando el ejecutable (*launcher*) de DualSPHysics correspondiente según el SO. DualSPHysics leerá el fichero de configuración de entrada XML e instanciará un objeto de tipo *JChronoObjects* cuando detecte la etiqueta *case.execution.special.**chrono***. Este objeto será el encargado de comunicarse con la librería DSPHChronoLib, mientras que DSPHChronoLib será la encargada de comunicarse con Chrono. Finalmente, tanto DualSPHysics como DSPHChronoLib guardarán determinados datos de la simulación en ficheros CSV.

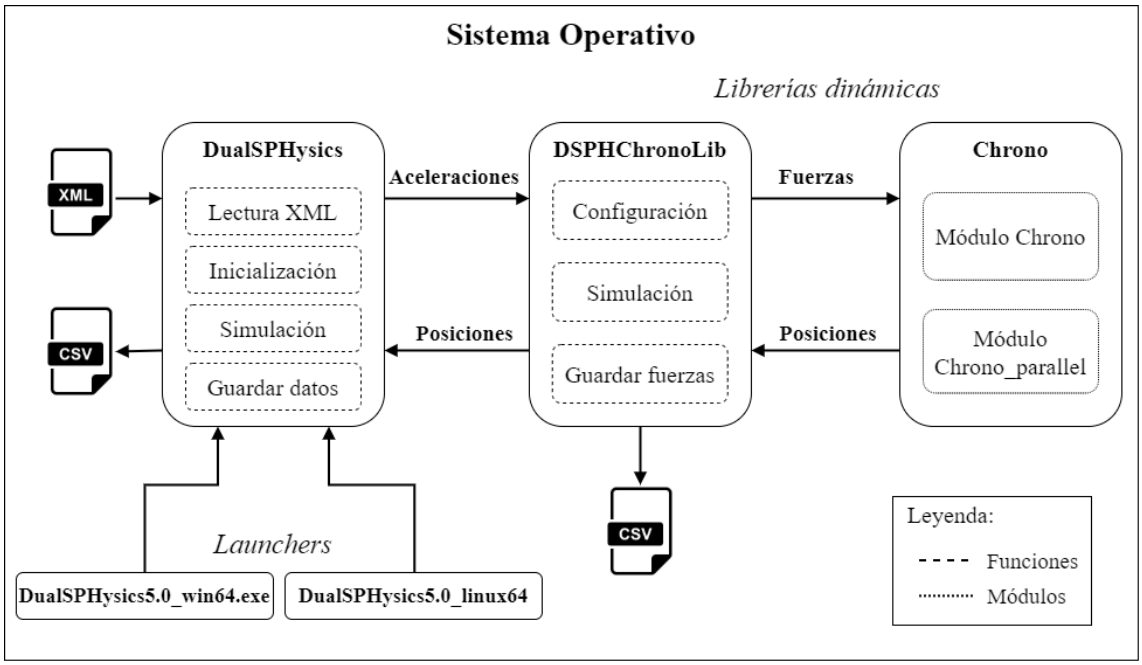

Figura 5. Arquitectura del acoplamiento

4 TECNOLOGÍAS Y HERRAMIENTAS

C++

Es un lenguaje de programación de propósito general, diseñado en 1979 por Bjarne Stroustrup [12]. La intención de su creación fue extender las características del lenguaje de programación C. Las características del lenguaje C++ permiten cuatro estilos de programación: programación procedural, abstracción de datos, programación orientada a objetos y programación genérica. Este lenguaje tiene un alto rendimiento al permitir hacer llamadas directamente al Sistema Operativo, es multiplataforma y permite acceso directo a la memoria (controlado por el usuario). Tiene una integración directa con el lenguaje ensamblador, permitiendo escribir directamente en dicho lenguaje. A pesar de los años que tiene, es un lenguaje moderno y actualizado. C++20 es la versión más reciente (2020), reemplazando la anterior especificación C++17 del año 2017.

DualSPHysics y Chrono están implementados en C++. Por esta razón también se utilizará este lenguaje para realizar la implementación asociada a este trabajo. Se utilizará la versión C++14 por motivos de compatibilidad con DualSPHysics y por estar soportado por el compilador Microsoft Visual C++ 2015.

GCC

GNU Compiler Collection (Colección de Compiladores GNU) [13]. Fue desarrollado por Richard Stallman, el fundador del Proyecto GNU. GCC es una colección de compiladores GNU que incluye interfaces para C++, entre otros lenguajes. En concreto, g++ es un comando de compilación para programas escritos en lenguaje C++ especialmente en plataformas Unix, aunque actualmente también da soporte a Windows. Se utilizará la versión GCC 10.2.0, lanzada en julio del año 2020.

GNU Make

Es una herramienta de gestión de dependencias de GNU [14]. Está diseñada para automatizar los pasos necesarios para la compilación de código fuente. Esta automatización se consigue mediante la definición de un fichero *makefile*, que por lo general se nombra *GNUmakefile*, *makefile* o *Makefile*. Un archivo *makefile* es un fichero de texto ordinario con una sintaxis muy concreta que Make puede entender. En este archivo se definen las relaciones entre el código fuente, los archivos intermedios y los ejecutables. Se escriben las dependencias, objetivos y los comandos para compilar archivos de código fuente y generar un ejecutable o librería compilada. Se utilizarán conjuntamente Make y GCC para la compilación del código fuente de la librería para plataformas Linux.

GDB

GDB o GNU Debugger [15] es el depurador estándar para el compilador GNU. Se utiliza en plataformas Unix y funciona para el lenguaje C++. Ofrece la posibilidad de ver lo que sucede dentro de otro programa mientras se ejecuta, o ver las causas de un posible bloqueo. GDB puede realizar varias tareas para ayudar a detectar errores. Por ejemplo: iniciar el programa especificando cualquier factor que pueda afectar a su comportamiento, hacer que el programa se detenga en condiciones específicas, examinar lo sucedido cuando el programa se detiene en circunstancias imprevisibles, etc. Es una herramienta muy útil para detectar posibles pérdidas de memoria o accesos a memoria no permitidos. Se utilizará la versión GDB 9.2, lanzada en mayo del año 2019.

CMake

CMake (*Cross-platform Make*) [16] es una herramienta multiplataforma de código abierto (*open source*) desarrollada para automatizar la construcción y generación de proyectos software. Es totalmente independiente de la plataforma y de los compiladores instalados en el equipo. CMake es capaz de generar proyectos para IDEs como Microsoft Visual Studio o generar ficheros *makefile* para hacer uso de Make. Permite compartir configuraciones entre los distintos proyectos, buscar librerías, ficheros de cabecera, etc. El proceso de construcción se controla mediante un fichero de texto plano denominado *CMakeLists.txt*, donde se definen las características que tendrá el proyecto.

DualSPHysics y Chrono también ofrecen un fichero *CMakeLists.txt* para utilizar CMake. Por ello, en este trabajo se utilizará CMake 3.16.2.

XML

XML (*eXtensible Markup Language*) [17] es un metalenguaje de marcado similar a HTML. Es una adaptación del SGML (*Standard Generalized Markup Language*). Permite la organización y etiquetado de documentos. No es un lenguaje en sí mismo, sino un sistema que permite definir lenguajes de acuerdo con las necesidades de cada uno. Ofrece la posibilidad de estructurar y representar datos. Es habitual que ciertos programas incluyan archivos de configuración con este formato.

DualSPHysics utiliza un fichero configuración de entrada XML. Por esta razón, también se utilizará este metalenguaje para la configuración de DSPHChronoLib y Chrono.

UML

El Lenguaje Unificado de Modelado (UML) [11] es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar artefactos de un sistema software. Captura decisiones y conocimiento sobre sistemas que deben ser construidos. Se usa para comprender, diseñar y controlar la información sobre dichos sistemas. UML capta la información sobre la estructura estática y el comportamiento dinámico del sistema. La estructura estática define los tipos de objetos importantes para un sistema y su implementación, así como las relaciones entre los objetos. El comportamiento dinámico define la historia de los objetos a lo largo del tiempo y la comunicación entre ellos para cumplir los objetivos.

En este proyecto se hará uso de los diagramas de clases para representar la estructura estática del sistema y los diagramas de secuencia para representar el comportamiento dinámico.

Visual Paradigm

Visual Paradigm (VP) [18] es una herramienta software diseñada para modelar los sistemas de información y gestionar los procesos de desarrollo que se van a llevar a cabo. Es compatible con lenguajes de modelado como el Lenguaje Unificado de Modelado (UML). Es una herramienta de pago, pero ofrece una versión *Community* gratuita. Se utilizará la versión 16.2 para la modelar software en este trabajo.

Microsoft Visual Studio

Microsoft Visual Studio [19] es un entorno de desarrollo integrado (*Integrated Development Environment*, IDE) para sistemas operativos Windows. Visual Studio ofrece herramientas como Microsoft Visual C++ que soporta el lenguaje de programación C++. Es un entorno de desarrollo que permite la creación de aplicaciones y librerías, ya que se pueden instalar compiladores para C++. También permite configurar el uso de librerías estáticas adicionales o librerías de enlace dinámico (DLL).

En este proyecto se utilizará Microsoft Visual Studio 2015 por motivos de compatibilidad, debido a que la última versión de DualSPHysics está compilada para Windows con Visual C++ 2015.

Chrono

Es la librería multifísica de código abierto [8] que se acoplará con DualSPHysics. El código base utilizado será el publicado en el repositorio oficial de Chrono (versión 4.0.0). En este trabajo se definieron una serie de objetivos que implican cambios en el código original. Por ello, el código será modificado para dotar al acoplamiento de nuevas funcionalidades no contempladas en la versión oficial de Chrono.

5 ESPECIFICACIÓN Y ANÁLISIS DE REQUISITOS

En este apartado se documenta la especificación y análisis de requisitos realizados en este trabajo dividido por iteraciones.

5.1 ITERACIÓN 0. ESTUDIO DEL PROBLEMA

Esta iteración pertenece a la fase de análisis. En ella ha realizado una especificación y análisis de requisitos inicial, dando lugar a los apartados INTRODUCCIÓN, OBJETIVOS, METODOLOGÍA, PLANIFICACIÓN, PUNTOS CRÍTICOS, que complementan al análisis documentado en este apartado.

Se realizará un acoplamiento entre el modelo DualSPHysics y la librería multifísica Chrono. Este acoplamiento se llevará a cabo mediante el desarrollo de una librería dinámica multiplataforma denominada DSPHChronoLib.

El nuevo acoplamiento deberá mantener las funcionalidades y el flujo de comunicación existentes anteriormente para minimizar los cambios en DualSPHysics y asegurar su correcto funcionamiento.

Será necesario mejorar el rendimiento de Chrono cuando se simulan un número elevado de objetos. Para ello, se añadirá el módulo *PARALLEL* y se permitirá seleccionar entre el modo de ejecución paralela y el de ejecución secuencial.

Se añadirán nuevas funcionalidades al acoplamiento:

- Colisiones elásticas (*SMooth Contacts*).
- Imponer un coeficiente de fricción cuando las colisiones están habilitadas.
- Nuevos *links*: muelle con fuerza de Coulomb (*Coulomb damping*) y polea (*pulley*).
- Variar los coeficientes de amortiguación (*damping*) y elasticidad (*stiffness*) para los *links* durante la ejecución.
- Salida de datos: Guardar la información intercambiada entre DualSPHysics y Chrono y las fuerzas lineales y angulares de los elementos de Chrono en CSV.

5.2 ITERACIÓN 3. ACOPLAMIENTO CON DSPHCHRONOLIB

En esta iteración se documenta el análisis correspondiente al desarrollo de la librería DSPHChronoLib y su acoplamiento con el modelo DualSPHysics.

DualSPHysics incluye un fichero de código fuente llamado *JChronoObjects.cpp/h* que se encarga de la lectura del fichero de configuración de entrada XML para definir los objetos de Chrono y sus características. Esta información se encuentra dentro de la etiqueta *case.execution.special.**chrono***. Se mantendrá esta jerarquía de etiquetas XML para configurar Chrono.

El fichero *JChronoObjects* también era el encargado de comunicarse con Chrono en la versión anterior. Se mantendrá este fichero como nexo entre DualSPHysics y DSPHChronoLib para la transferencia de información. Las nuevas funcionalidades relacionadas con Chrono que impliquen cambios en DualSPHysics, se realizarán principalmente en *JChronoObjects*.

DSPHChronoLib será una librería dinámica multiplataforma que deberá funcionar en Windows y Linux. Los proyectos se generarán con la herramienta CMake. Por ello se implementará un fichero *CMakeLists.txt*, que permita generar soluciones de Microsoft Visual Studio en Windows y ficheros *Makefile* en Linux.

DSPHChronoLib tendrá un fichero *JChronoData.cpp/h*. En este fichero se implementará la clase *JChronoData* que almacenará los datos leídos por *JChronoObjects* en el fichero de configuración de entrada XML. En *JChronoData.cpp/h* también se implementarán las clases correspondientes a cada tipo de objeto y *link* disponible en el acoplamiento.

DSPHChronoLib tendrá dos funciones para la comunicación principal con Chrono.

- **DSPHChronoLib::Config** : *void*. En esta función se crearán, parametrizarán y añadirán al sistema de Chrono todos los elementos definidos en el fichero de configuración de entrada XML. Se ejecutará una única vez durante la simulación.
- **DSPHChronoLib::RunChrono** : *void*. Esta función será la encargada de indicarle a Chrono que realice la simulación de un determinado intervalo de tiempo. Se ejecutará en cada paso de tiempo de la simulación con DualSPHysics hasta que esta finalice.

5.3 ITERACIÓN 4. ACOPLAMIENTO CON CHRONO

En esta iteración se realizará el acoplamiento entre DSPHChronoLib y Chrono y se añadirán las funcionalidades existentes en el acoplamiento antiguo, documentadas a continuación:

- Objetos rígidos: flotantes y fijos.
- *Links*: muelle lineal (*linear spring*), bisagra (*hinge*), articulación esférica (*joint spheric*), articulación punto-línea (*joint point-line*).
- Colisiones rígidas (*Non-Smooth Contacts*).
- Generación de ficheros VTK para la visualización de los elementos.

Se utilizará la clase *ChSystem* de Chrono para realizar las simulaciones de la librería. Este objeto se inicializará desde el constructor de la clase DSPHChronoLib.

En la función ***DSPHChronoLib::Config*** se implementará el acceso a cada objeto y *link* definido en *JChronoData*. Se leerá la información y se crearán e inicializarán los objetos de Chrono a partir de estos datos. Una vez configurados, se añadirán al sistema de Chrono (*ChSystem*).

En la función ***DSPHChronoLib::RunChrono*** se calcularán y aplicarán las fuerzas de cada objeto de Chrono a partir de las aceleraciones lineales y angulares obtenidas por DualSPHysics. Posteriormente, se llamará a la función ***ChSystem::DoFrameDynamics(timestep)*** para que Chrono resuelva el desplazamiento de los objetos. Al finalizar, se actualizarán las nuevas posiciones de los objetos en *JChronoData* para que DualSPHysics aplique el nuevo estado de cada elemento calculado por Chrono.

5.4 ITERACIÓN 5. SALIDA DE DATOS

En esta iteración se realizará la exportación de datos en formato CSV. Se generarán dos tipos de ficheros CSV:

- Tipo 1. Almacenará la información intercambiada entre DualSPHysics y Chrono. Este fichero se denominará *ChronoExchange_mkbound_XX.csv*, donde XX es el identificador del objeto flotante (*mkbound*).
- Tipo 2. Almacenará las fuerzas lineales y angulares utilizadas por Chrono para mover los objetos. Este tipo de fichero se dividirá en dos CSV: uno para objetos rígidos (*ChronoBody_forces.csv*) y otro para *links* (*ChronoLink_forces.csv*).

La generación de ambos tipos de ficheros se configurará usando la opción *savedata* dentro del bloque *chrono*, donde se indica la frecuencia de guardado de datos, tal como se puede observar en la Figura 6.

```
<chrono>
    <savedata value="0.01" comment="Saves CSV with data exchange for
                                    each time interval (0=all steps)"/>

</chrono>
```

Figura 6. Opción para habilitar y configurar la exportación de datos en ficheros CSV

5.5 ITERACIÓN 6. MÓDULO PARALELO

En esta iteración se reducirá el tiempo de ejecución en Chrono. Para ello se utilizará un módulo de computación paralela existente en Chrono, denominado *PARALLEL*.

La opción de utilizar la computación paralela de Chrono solamente estará disponible cuando las colisiones estén activadas, ya que es cuando se necesita un mayor rendimiento.

Se realizará una especialización de la clase principal de DSPHChronoLib para favorecer la reutilización de código, evitar la duplicidad y facilitar el mantenimiento y la ampliación de la librería. Se implementarán dos clases que hereden de DSPHChronoLib:

- DSPHChronoLibSC. Permitirá las ejecuciones secuenciales (*Single Core*).
- DSPHChronoLibMC. Permitirá las ejecuciones paralelas mediante el módulo *PARALLEL* (*Multi Core*).

Se podrá seleccionar el modo de ejecución indicando el número de hilos (*threads*) en el fichero de configuración de entrada XML con la etiqueta *ompthreads*. En caso de seleccionar el valor 0, se realizará la ejecución paralela creando tantos *threads* como número de núcleos tenga el procesador. En caso contrario, se utilizará el número de *threads* indicados. Para ejecuciones secuenciales se introducirá un 1 y será la opción por defecto. La Figura 7 muestra cómo se podrá definir esta opción.

```
<!-- Single-Core or Multi-Core -->
<collision activate="true">
    <ompthreads value="0" comment="Number of threads by host
                                   for parallel execution.
                                   0:Multi-Core, 1:Single-Core (default=1)" />
</collision>
```

Figura 7. Selección del modo de ejecución en Chrono

Se realizarán modificaciones en el fichero *CMakeLists.txt* para acceder a la configuración de CMake de Chrono y conocer los módulos que están activados y generados. De esta forma se podrá habilitar/deshabilitar el uso de la clase DSPHChronoLibMC en función de si el módulo *PARALLEL* está activado/desactivado.

5.6 ITERACIÓN 7. COLISIONES

En esta iteración se añadirán nuevas funcionalidades relacionadas con las colisiones entre objetos. Chrono permite simular dos tipos de colisiones: rígidas (*Non-Smooth Contacts*, NSC) y elásticas (*SMooth Contacts*, SMC). En el acoplamiento antiguo solamente estaban disponibles las colisiones NSC.

Es necesario definir las propiedades del material que compone la superficie de cada objeto para habilitar las colisiones. Los coeficientes que definirán el tipo de material son los siguientes:

- Restitución (*restitution*, C_R). Ratio que mide la relación entre la velocidad relativa final e inicial entre dos objetos después de la colisión.
- Fricción o rozamiento (*friction*, μ). Ratio de la fuerza que ofrece una resistencia al movimiento entre dos objetos que se deslizan entre sí.
  - Fricción estática (*static friction*, μ_s). Resistencia entre dos objetos que no están en movimiento.
  - Fricción cinética (*kinetic friction*, μ_k). Resistencia entre objetos en movimiento (*sliding*).

Las colisiones SMC utilizan dos coeficientes más para representar el comportamiento del material. Será necesario leer del fichero de configuración de entrada XML los siguientes parámetros:

- Módulo de elasticidad o Young (*Young's modulus*, E). Parámetro que mide la rigidez de materiales sólidos.
- Coeficiente de Poisson (*Poisson's ratio*, n). Mide la deformación de un material en la dirección perpendicular a la dirección donde se aplica un estiramiento longitudinal.

Se seleccionará el tipo de colisión desde el fichero de configuración de entrada XML mediante la etiqueta *chrono.collision.**contactmethod***, tal como se puede observar en la Figura 8.

```
<chrono>
    <collision activate="true">
        <contactmethod value="0" comment="Contact method type.
                                          0:NSC (Non Smooth Contacts),
                                          1:SMC (SMooth Contacts).
                                        (default=0)" />
    </collision>
</chrono>
```

Figura 8. Selección del método de contacto

Cuando dos objetos colisionan entre sí, cada uno tiene un coeficiente de fricción asociado a su material (*fric1*, *fric2*). Chrono resuelve la colisión utilizando el *min(fric1, fric2)*. Se añadirá una opción para imponer un coeficiente de fricción de un objeto frente a los demás cuando colisionen (*imposefric*="*true*") y evitar que Chrono lo calcule internamente. En la Figura 9 se puede observar cómo será el uso de esta opción.

```
<chrono>
    <bodyfixed id="domain" mkbound="10" imposefric="true"/>
    <bodyfloating id="box" mkbound="51"/>
</chrono>
```

Figura 9. Uso de la opción de imponer coeficientes de fricción

5.7 ITERACIÓN 8. ENLACES ENTRE OBJETOS

En esta iteración se añadirán varias funcionalidades relacionadas con los enlaces entre objetos o *links* de Chrono. Los *links* permiten conectar dos objetos entre sí añadiendo restricciones físicas.

Se añadirán dos tipos de *links*: muelle lineal con fuerza de Coulomb (*Coulomb damping*) y polea (*pulley*).

El *Coulomb damping* no existe en Chrono, por lo que será necesario realizar cambios en su código original. Este *link* estará basado en un muelle lineal (*linear spring*).

El *linear spring* tiene un sistema de toma de fuerza (*power take-off*, PTO) utilizado en casos de aplicación de generación de energía. La fuerza PTO (F_{PTO}) de este *link* se calcula según la ecuación (1).

$$F_{PTO} = k \cdot deform + c \cdot vel.z \tag{1}$$

donde *k* [N/m] es la elasticidad (*stiffness)*, *c* [Ns/m] es la amortiguación (*damping*), *deform* [m] es la diferencia entre la longitud actual del muelle y el tamaño del muelle en equilibrio y *vel.z* [m/s] es la velocidad en el eje Z.

En el caso de *Coulomb damping*, se modificará la formulación y el cálculo de F_{PTO}. La nueva formulación se muestra en la ecuación (2).

$$F_{PTO} = -\text{sign}(vel.z) \cdot F_b \tag{2}$$

donde *vel.z* [m/s] es la velocidad en el eje Z y F_b es la fuerza de Coulomb [N].

Se implementará la lectura de este nuevo tipo de *link* en el fichero de configuración de entrada XML. Se definirá indicando los dos objetos conectados, las posiciones de la conexión, la longitud del muelle en equilibrio y la fuerza de Coulomb, tal como se muestra en la Figura 10.

```
<!-- Coulomb Damping bewteen two bodies -->
<link_coulombdamping  idbody1="domain" idbody2="box1">
    <point_fb1 x="1.0" y="1.0" z="1.0" comment="Point in body 1" />
    <point_fb2 x="2.0" y="2.0" z="2.0" comment="Point in body 2" />
    <rest_length value="5.0" comment="Spring equilibrium length [m]" />
    <damping value="100" comment="Coulomb force [N]" />
</link_coulombdamping>
```

Figura 10. Definición de un link Coulomb damping

El *link Pulley* está disponible en Chrono y permite conectar dos objetos aplicando una restricción de giro. Uno de los objetos (*sphere1*) será el "maestro" e impondrá un giro sobre el otro objeto (*sphere2*) que será el "esclavo". Se implementará la lectura de este tipo de *link* en el fichero de configuración de entrada XML. Se definirá indicando los objetos conectados, el punto de rotación, el vector de rotación y los radios de los dos objetos, tal como se muestra en la Figura 11.

```
<!-- Pulley bewteen two bodies -->
<link_pulley idbody1="sphere1" idbody2="sphere2">
    <rotpoint x="1.0" y="0" z="1.2" comment="Point for rotation" />
    <rotvector x="0.0" y="1.0" z="0.0" comment="Rotation axis of the body" />
    <radius value="0.3" comment="Radius of idbody1"/>
    <radius2 value="0.05" comment="Radius of idbody2"/>
</link_pulley>
```

Figura 11. Definición de un link Pulley

Se añadirá la posibilidad de variar en tiempo de ejecución los coeficientes de elasticidad (*stiffness*) y amortiguación (*damping*) de los *links* que admitan estos parámetros: muelle lineal (*linear spring*), bisagra (*hinge*), articulación esférica (*joint spheric*), articulación punto-línea (*joint point-line*). Se realizará de dos formas:

- Desde un fichero externo CSV. Se introducirá el nombre del fichero CSV en el atributo *file* de cada coeficiente. El fichero CSV tendrá una estructura de 2 columnas (tiempo y coeficiente) y cada fila tendrá el valor del coeficiente a aplicar en cada tiempo de simulación.

```
<!-- Spring with variable stiffness and damping from an external file -->
<link_linearspring idbody1="domain" idbody2="box1">
    <point_fb1 x="1.0" y="1.0" z="1.0" comment="Point in body 1" />
    <point_fb2 x="2.0" y="2.0" z="2.0" comment="Point in body 2" />
    <damping file="CaseExternalDamping.csv" comment="Damping [Ns/m]"/>
    <stiffness file="CaseExternalStiffness.csv" comment="Stiffness [N/m]"/>
    <rest_length value="5.0" comment="Spring equilibrium length [m]" />
</link_linearspring>
```

Figura 12. Coeficientes variables de damping y stiffness desde CSV

- Embebido en el fichero de configuración de entrada XML. Dentro de las etiquetas *damping* y *stiffness* se introducirá un valor de *c* o *k* respectivamente y el tiempo de simulación en el que se aplicarán los coeficientes.

```
<!-- Spring with variable stiffness and damping embedded in XML -->
<link_linearspring idbody1="domain" idbody2="box1">
    <point_fb1 x="1.0" y="1.0" z="1.0" comment="Point in body 1" />
    <point_fb2 x="2.0" y="2.0" z="2.0" comment="Point in body 2" />
    <damping comment="Damping [Ns/m]">
        <c time="0" value="0"/>
        <c time="5" value="0"/>
        <c time="5" value="500"/>
        <c time="10" value="500"/>
    </damping>
    <stiffness comment="Stiffness [N/m]">
        <k time="0" value="0"/>
        <k time="5" value="0"/>
        <k time="5" value="100"/>
        <k time="10" value="100"/>
    </stiffness>
    <rest_length value="5.0" comment="Spring equilibrium length [m]" />
</link_linearspring>
```

Figura 13. Coeficientes variables de stiffness y damping embebido en el XML

6 DISEÑO DEL SOFTWARE

En este apartado se documenta el diseño de software estático y dinámico realizado a lo largo del trabajo. El diseño estático se ha representado mediante un diagrama de clases (artefacto UML), que es el resultado de la Iteración 1. El diseño dinámico se ha representado con diagramas de secuencia (artefactos UML), que son el resultado de las tareas de diseño de cada iteración de la fase de implementación. Se ha decido no incluir las funciones *getter* y *setter* de los atributos con accesibilidad privada o protegida, ni todas las funciones y atributos de todas las clases debido a su extensión. Solamente se contemplan los más representativos que intervienen en el flujo de comunicación del acoplamiento implementado.

6.1 DISEÑO ESTÁTICO

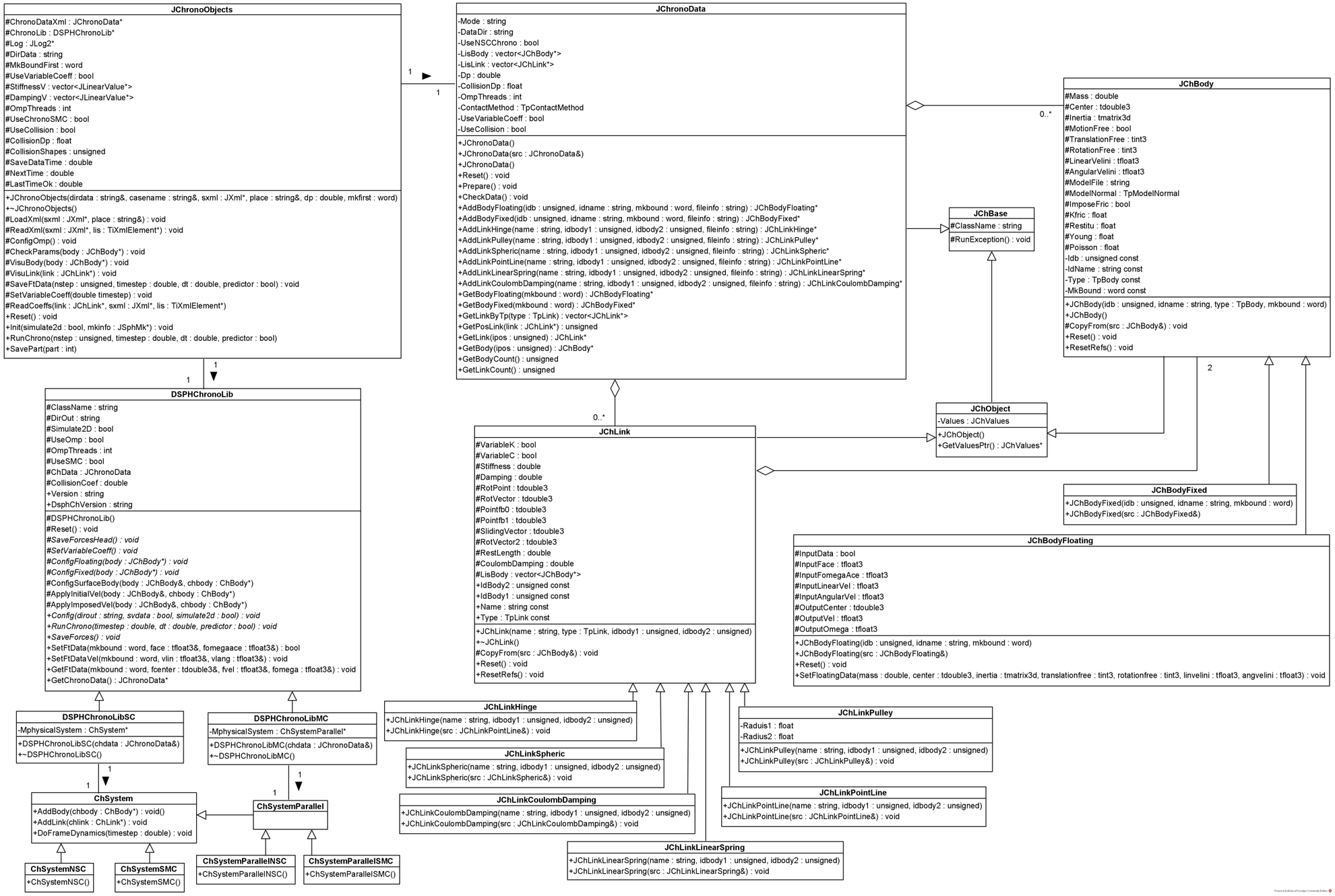

Figura 14. Diagrama de Clases

6.2 DISEÑO DINÁMICO

En cada diagrama de secuencia se añade una nota en la línea de vida de la clase que comienza la secuencia de ejecución. En esta nota se indica la función donde se inicia el flujo de ejecución que se muestra en cada diagrama.

6.2.1 ITERACIÓN 3. ACOPLAMIENTO CON DSPHCHRONOLIB

El diagrama de la Figura 15 muestra el flujo de comunicación entre DualSPHysics y DSPHChronoLib. Se ha definido un bloque *sd* que engloba las funciones *JChronoObjects::Init()* y *JChronoObjects::VisuConfig()* para poder referenciarlo desde otro diagrama y evitar duplicidad.

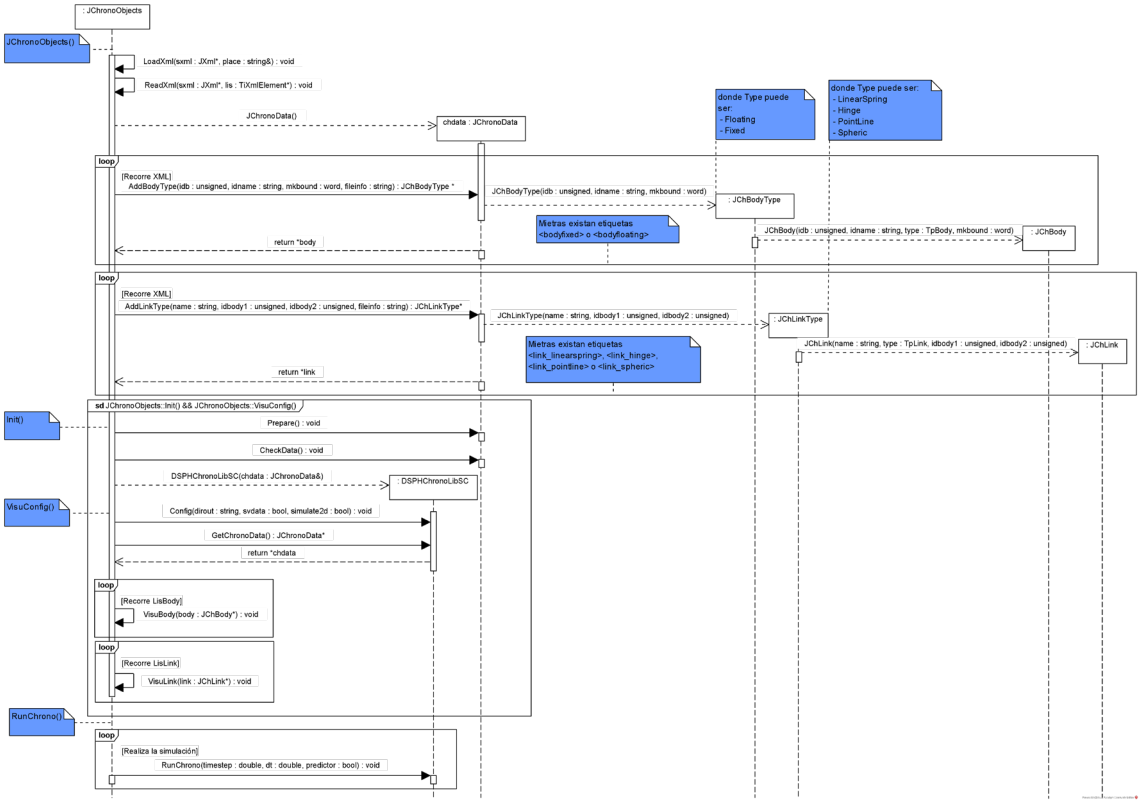

Figura 15. Diagrama de Secuencia del acoplamiento con DSPHChronoLib

6.2.2 ITERACIÓN 4. ACOPLAMIENTO CON CHRONO

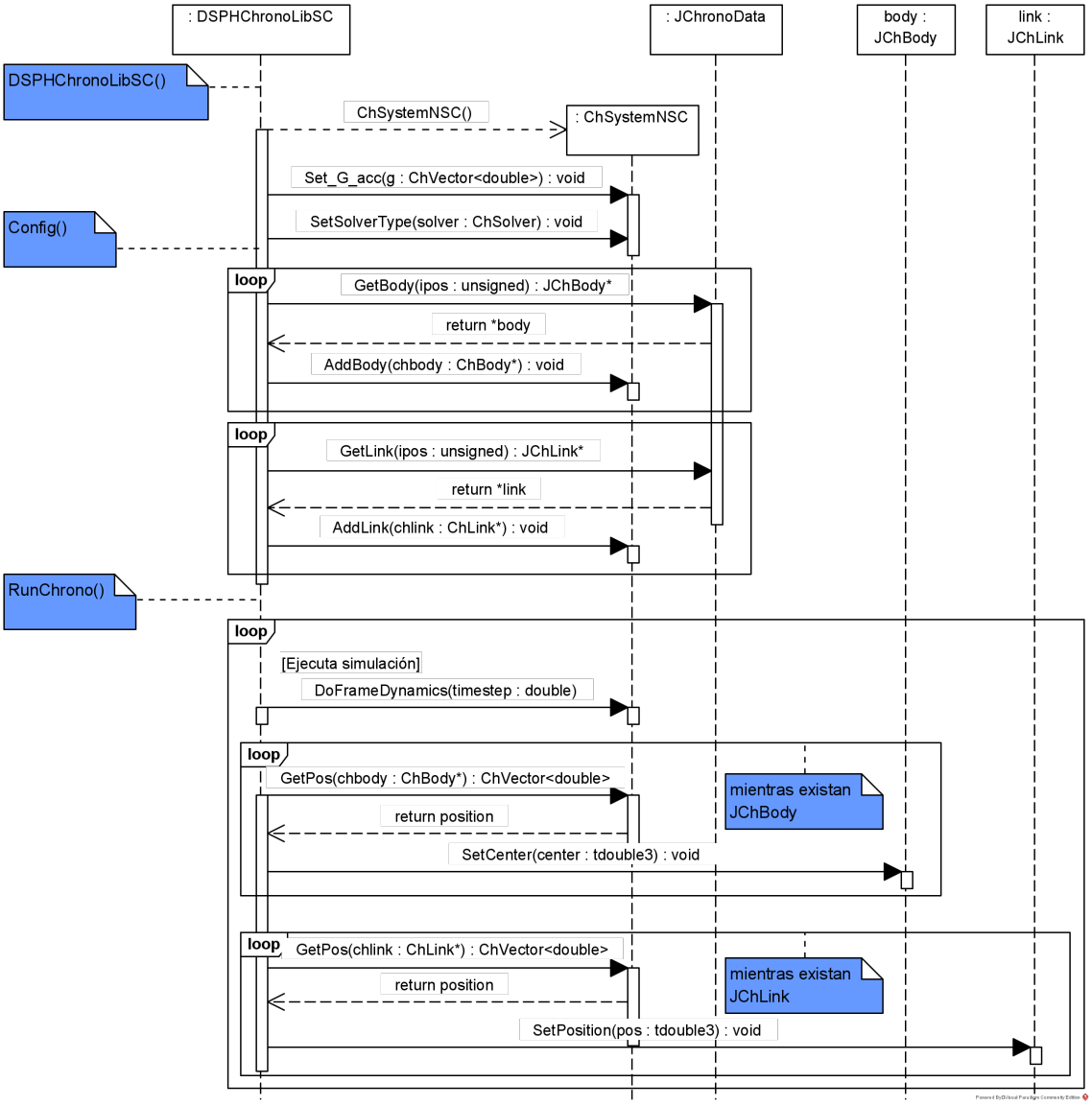

Figura 16. Diagrama de Secuencia del acoplamiento con Chrono

6.2.3 ITERACIÓN 5. SALIDA DE DATOS

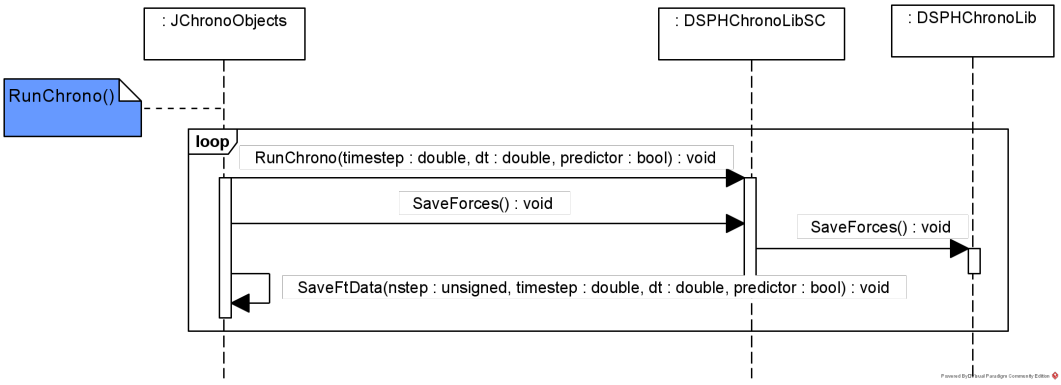

Figura 17. Diagrama de Secuencia de la salida de datos

6.2.4 ITERACIÓN 6. MÓDULO PARALELO

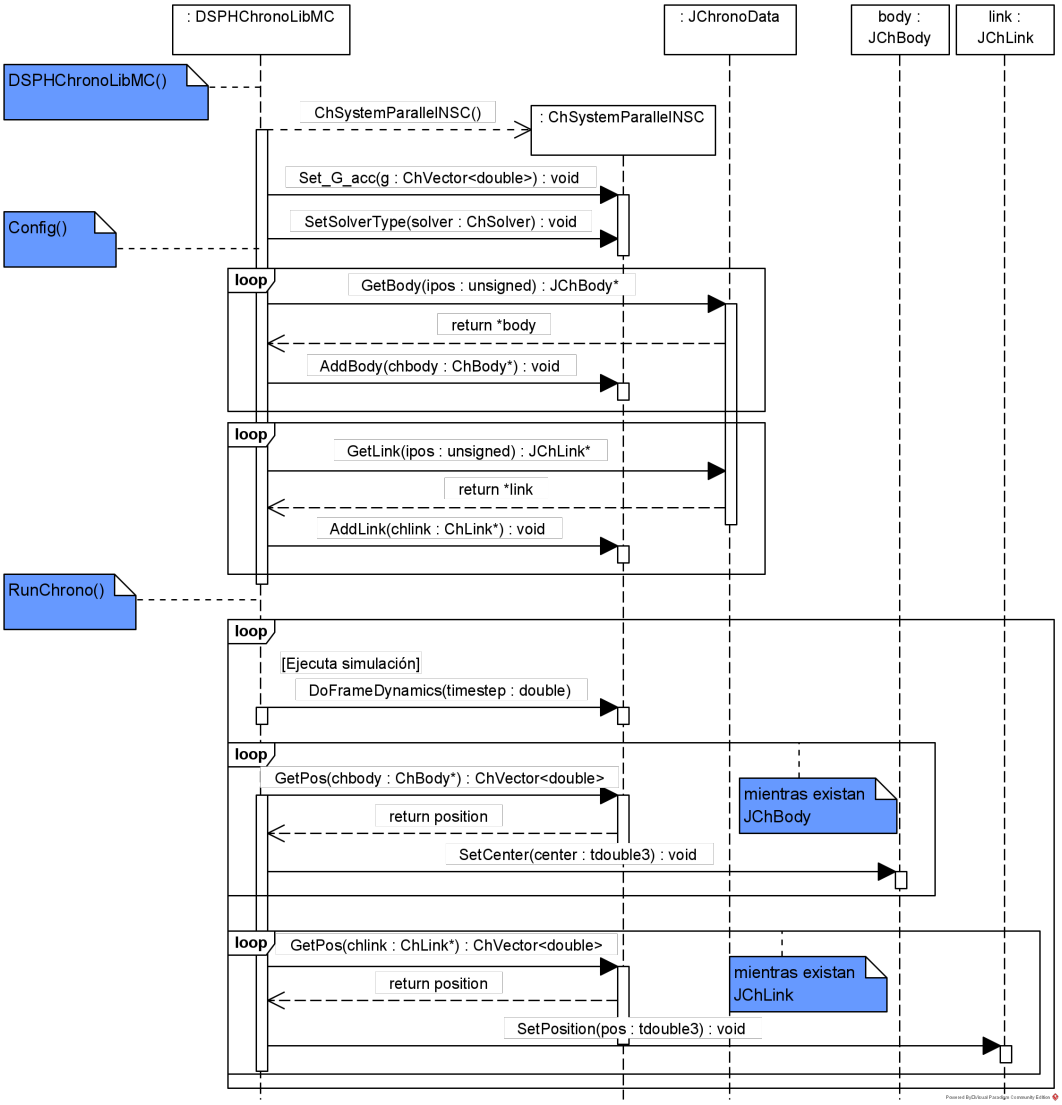

Figura 18. Diagrama de Secuencia del módulo paralelo

6.2.5 ITERACIÓN 7. COLISIONES

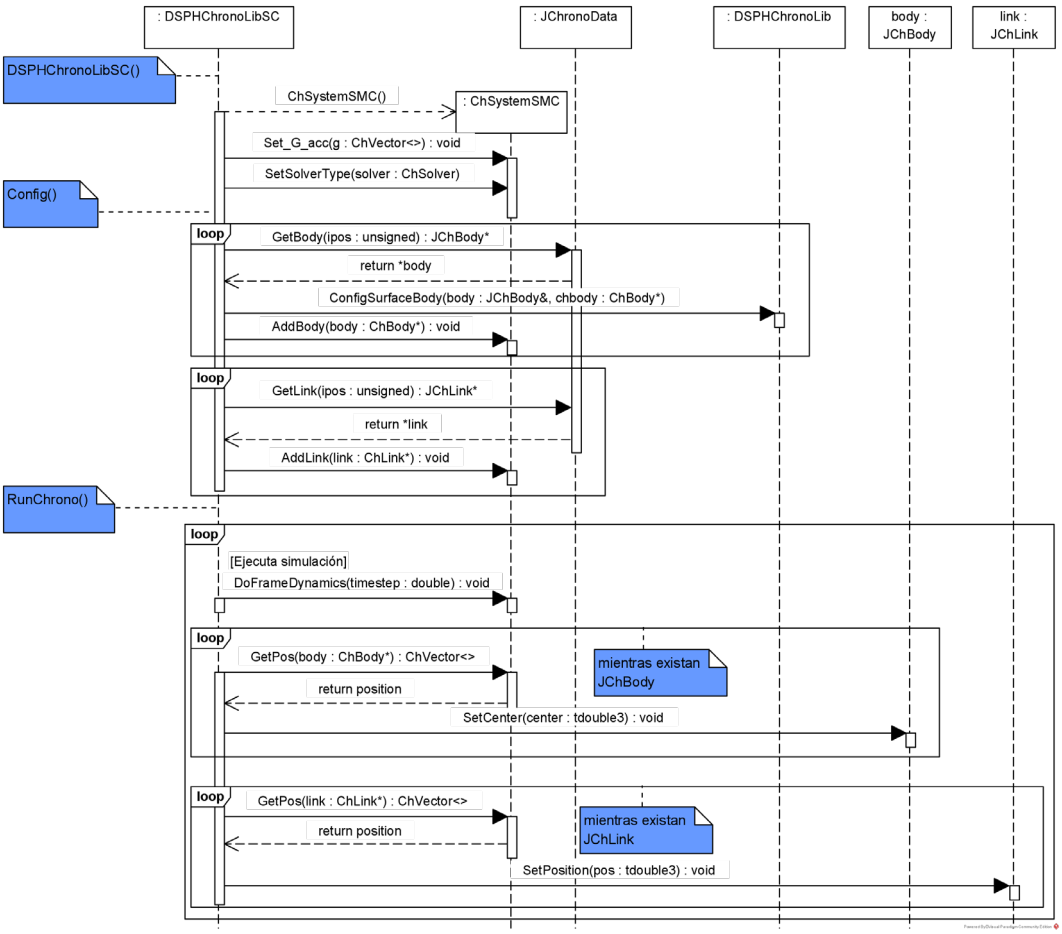

Figura 19. Diagrama de Secuencia de las colisiones

6.2.6 ITERACIÓN 8. ENLACES ENTRE OBJETOS

En el diagrama de secuencia de la Figura 20 se puede observar un bloque *ref* que hace referencia al fragmento denotado con *sd*, del diagrama de la Figura 15.

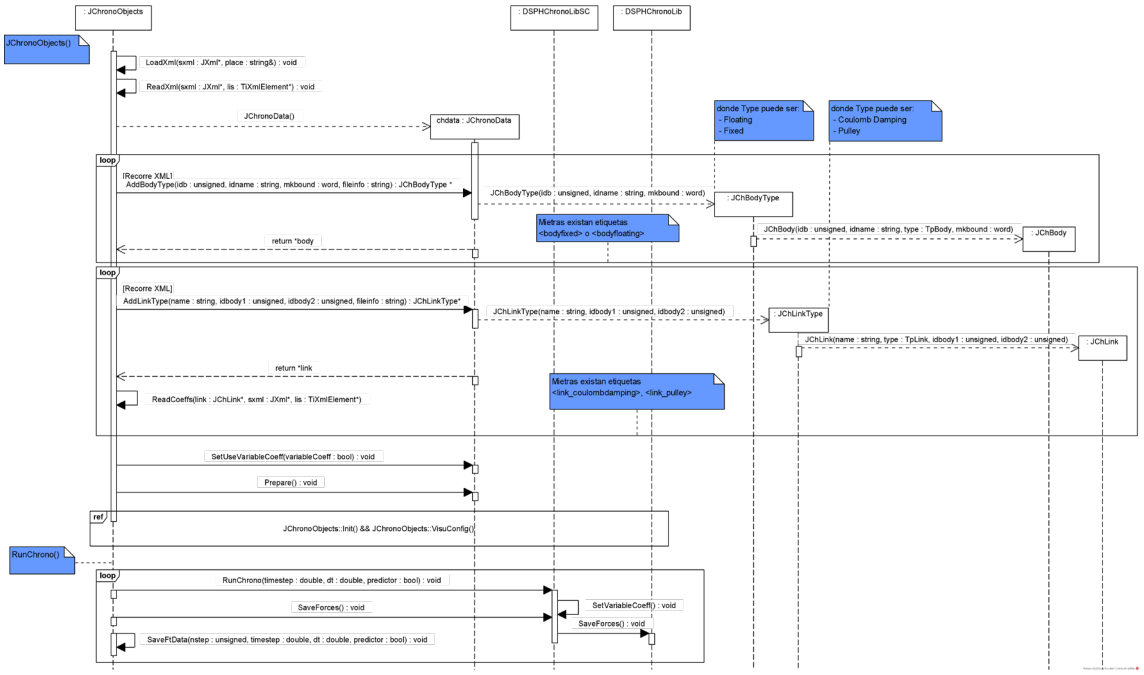

Figura 20. Diagrama de Secuencia de los enlaces entre objetos

7 GESTIÓN DE DATOS E INFORMACIÓN

En este apartado se documenta la entrada y salida de datos del acoplamiento. La entrada de datos se realiza mediante un fichero XML. La configuración Chrono se introducirá dentro de un bloque *case.execution.special.**chrono*** que será leído por la clase *JChronoObjects* de DualSPHysics. La Figura 21 muestra la estructura general del bloque ***chrono***.

- **savedata**. Habilita el guardado de fuerzas en ficheros CSV.
- **collision**. Habilita las colisiones.
  - **distancedp**. Superposición permitida entre objetos cuando colisionan.
  - **ompthreads**. Selecciona número de hilos con los que se ejecutará Chrono.
  - **contactmethod**. Selecciona el tipo de colisiones (NSC o SMC).
- **bodyfixed**. Objeto rígido fijo.
- **bodyfloating**. Objeto flotante.
- **link_linearspring**. Muelle lineal.
- **link_coulombdamping**. Muelle con fuerza de Coulomb.
- **link_hinge**. Bisagra.
- **link_pointline**. Articulación punto-línea.
- **link_spheric**. Articulación esférica.
- **link_pulley**. Polea.

```
<chrono>
    <savedata/>
    <collision>
        <distancedp/>
        <ompthreads/>
        <contactmethod/>
    </collision>
    <bodyfixed/>
    <bodyfloating/>
    <link_linearspring/>
    <link_coulombdamping/>
    <link_hinge/>
    <link_pointline/>
    <link_spheric/>
    <link_pulley/>
</chrono>
```

Figura 21. Plantilla de configuración de Chrono

La salida de datos se realiza mediante dos tipos de ficheros CSV:

- Tipo 1. Información intercambiada entre DualSPHysics y Chrono de los objetos flotantes. Este fichero tendrá 18 columnas, tal como se puede observar en la Tabla 4.

nstep	time [s]	dt [s]	face.x [m/s2]	face.y [m/s2]	face.z [m/s2]	fomegaace.x [rad/s2]	fomegaace.y [rad/s2]	fomegaace.z [rad/s2]
fvel.x [m/s]	fvel.y [m/s]	fvel.z [m/s]	fcenter.x [m]	fcenter.y [m]	fcenter.z [m]	fomega.x [rad/s]	fomega.y [rad/s]	fomega.z [rad/s]

Tabla 4. Estructura del fichero CSV de datos intercambiados entre DualSPHysics y Chrono

Donde:

- **nstep**. Número de pasos de cálculo realizados en la simulación.
- **time**. Tiempo de simulación.
- **dt**. Paso de tiempo.
- **face [x y z]**. Aceleración lineal en los 3 ejes.
- **fomegaace [x y z]**. Aceleración angular en los 3 ejes.
- **fvel [x y z]**. Velocidad lineal en los 3 ejes.
- **fcenter [x y z]**. Posición del centro de masas en los 3 ejes.
- **fomega [x y z]**. Velocidad angular en los 3 ejes.

- Tipo 2. Fuerzas utilizadas por Chrono para desplazar los objetos. Este tipo de fichero se dividirá en dos CSV, uno de ellos almacenará todos los objetos rígidos (*Body*) y el otro almacenará los enlaces (*Link*). Cada fichero tendrá 6 columnas por cada objeto o *link* creado en Chrono, además de una columna de tiempo. La Tabla 5 muestra cómo será la estructura de estos ficheros.

time [s]	[Link|Body]_*name*_fx [N]	fy [N]	fz [N]	mx [Nm]	my [Nm]	mz [Nm]

Tabla 5. Estructura del fichero CSV de fuerzas

Donde:

- **time**. Tiempo de simulación.
- **f [x y z]**. Fuerza lineal en los 3 ejes.
- **m [x y z]**. Fuerza angular en los 3 ejes.

8 PRUEBAS LLEVADAS A CABO

En este apartado se documentan las pruebas más representativas llevadas a cabo en cada una de las iteraciones de la fase de implementación y las pruebas globales. Las pruebas realizadas sirven para validar las iteraciones que agregan funcionalidades.

8.1 ITERACIÓN 3. ACOPLAMIENTO CON DSPHCHRONOLIB

En esta iteración se valida el acoplamiento entre DualSPHysics y DSPHChronoLib asegurando que la comunicación entre ambos se realiza correctamente.

Se define un caso mediante el fichero de configuración de entrada XML. El caso está formado por un objeto rígido (*Bottom*), dos objetos flotantes (*Floater y FloaterLink*) y un enlace tipo muelle lineal (*link_linearspring*) que conecta al *FloaterLink* con el *Bottom*. La estructura del fichero de configuración XML se puede observar en la Figura 22.

```
<chrono>
    <bodyfixed id="Bottom" mkbound="0" />
    <bodyfloating id="FloaterLink" mkbound="50" />
    <bodyfloating id="Floater" mkbound="51" />
    <link_linearspring idbody1="Bottom" idbody2="FloaterLink">
        <point_fb1 x="1.15" y="0.0" z="0.0" comment="Point in body 1" />
        <point_fb2 x="1.15" y="0.0" z="0.325" comment="Point in body 2" />
        <stiffness value="100.0" comment="Stiffness [N/m]" />
        <damping value="500.0" comment="Damping [Ns/m]" />
        <rest_length value="0.325" comment="Spring equilibrium length [m]" />
    </link_linearspring>
</chrono>
```

Figura 22. Configuración del caso de validación del acoplamiento con DSPHChronoLib

Se ejecuta el caso y se comprueba que DualSPHysics leyó y envió correctamente la información del fichero de configuración XML a DSPHChronoLib. La Figura 23 muestra un fragmento de la salida por consola de DualSPHysics durante la ejecución de este caso, donde puede verse un resumen de los elementos de Chrono configurados.

```
Seleccionar C:\WINDOWS\system32\cmd.exe

Chrono Objects configuration:
  Data directory....: [Test_coupling_DSPHChronoLib_out/chrono_objs/]
  Collisions........: False
  Bodies............: 3
  Links.............: 1
  Body_0000 "Bottom" -  type: Fixed
    MkBound......: 0
    Links......: 1
      Link_Bottom_FloaterLink
  Body_0001 "FloaterLink" -  type: Floating
    MkBound......: 50
    Mass.........: 26.35
    Center.......: (1.15,0,0.4)
    Inertia......: (0.06324,0.27404,0.2108) (xx,yy,zz)
    Inertia......: (0,0,2.51535e-18) (xy,yz,xz)
    Links......: 1
      Link_Bottom_FloaterLink
  Body_0002 "Floater" -  type: Floating
    MkBound......: 51
    Mass.........: 26.35
    Center.......: (2.15,0,0.4)
    Inertia......: (0.06324,0.27404,0.2108) (xx,yy,zz)
    Inertia......: (0,0,-5.20417e-19) (xy,yz,xz)
    MotionFree...: Translation:(0,0,1) Rotation:(0,0,0)
  Link "Link_Bottom_FloaterLink" -  type: LinearSpring
    Point Body 1..: (1.15,0,0)
    Point Body 2..: (1.15,0,0.325)
    Rest length...: 0.325
    Stiffness.....: 100
    Damping.......: 500
    Bodies........: 2
      Body_0000 "Bottom"
      Body_0001 "FloaterLink"
```

Figura 23. Salida por consola del caso de validación del acoplamiento con DSPHChronoLib

La lectura y configuración de cada uno de los elementos se realiza correctamente, por ello esta funcionalidad queda validada e integrada.

8.2 ITERACIÓN 4. ACOPLAMIENTO CON CHRONO

En esta iteración se valida el acoplamiento entre DualSPHysics y Chrono. Se define un caso para realizar una simulación y probar el acoplamiento. El caso está formado por un objeto fijo (*Bottom*), dos objetos flotantes (*Floaters*) y un enlace de muelle lineal (*link_linearspring*) que conecta a otro objeto flotante (*FloaterLink*) con el *Bottom*. Se habilita el uso de colisiones NSC y la generación de ficheros VTK para la visualización de la simulación. También se define el fluido y un pistón para la generación de oleaje en DualSPHysics. La estructura del fichero de configuración XML dedicada a Chrono se puede observar en la Figura 24.

```
<chrono>
    <collision activate="true">
        <distancedp value="0.5" comment="Allowed collision overlap according Dp (default=0.5)" />
    </collision>
    <bodyfixed id="Bottom" mkbound="0" modelfile="AutoActual" modelnormal="invert" />
    <bodyfloating id="Floaters" mkbound="51-52" modelfile="AutoActual"/>
    <bodyfloating id="FloaterLink" mkbound="50" modelfile="AutoActual"/>
    <link_linearspring idbody1="Bottom" idbody2="FloaterLink">
        <point_fb1 x="1.15" y="0.0" z="0.0" comment="Point in body 1" />
        <point_fb2 x="1.15" y="0.0" z="0.325" comment="Point in body 2" />
        <stiffness value="100.0" comment="Stiffness [N/m]" />
        <damping value="500.0" comment="Damping [Ns/m]" />
        <rest_length value="0.325" comment="Spring equilibrium length [m]" />
        <savevtk>
            <nside value="16" comment="number of sections for each revolution. (default=16)" />
            <radius value="5.0" comment="spring radius (default=3)" />
            <length value="3.0" comment="length for each revolution (default=1)" />
        </savevtk>
    </link_linearspring>
</chrono>
```

Figura 24. Configuración del caso de validación del acoplamiento con Chrono

La Figura 25 muestra distintos instantes de la simulación, donde los objetos flotantes de color azul y rojo (*Floaters*) se desplazan de acuerdo con el movimiento del fluido. Sin embargo, el muelle lineal restringe el movimiento del objeto flotante naranja (*FloaterLink*), evitando que se desplace lateralmente. En la segunda imagen se puede observar que los *Floaters* colisionan entre sí, garantizando que las colisiones están activadas y funcionando correctamente.

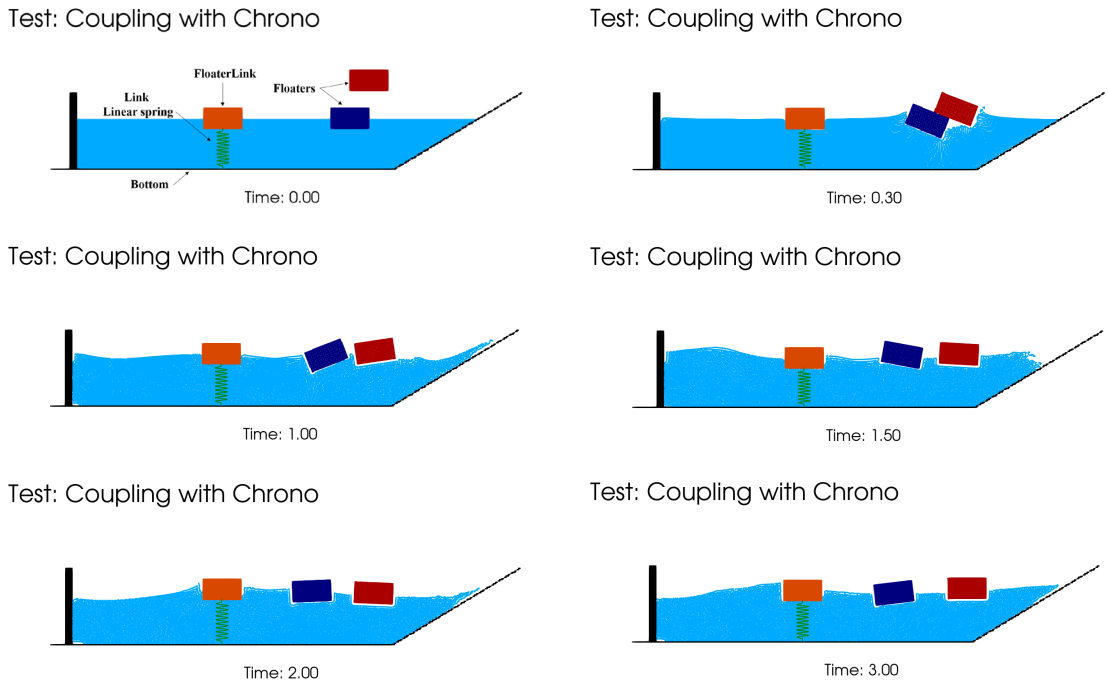

Figura 25. Secuencia de simulación para la validación del acoplamiento con Chrono

El resultado de la simulación demuestra que el acoplamiento con Chrono está funcionando correctamente, por lo tanto, esta iteración queda validada e integrada.

8.3 ITERACIÓN 5. SALIDA DE DATOS

En esta iteración se valida la salida de datos mediante la generación de ficheros CSV. Se utilizará el caso definido para la validación de la iteración anterior (8.2 ITERACIÓN 4. ACOPLAMIENTO CON CHRONO), al que se le añade la opción para la generación de ficheros CSV (*savedata*) con una frecuencia de 0.05 segundos. La Figura 26 muestra la estructura del fichero de configuración XML.

```
<chrono>
    <savedata value="0.05" comment="Saves CSV with data exchange for each time interval (0=all steps)" />
    <collision activate="true">
        <distancedp value="0.5" comment="Allowed collision overlap according Dp (default=0.5)" />
    </collision>
    <bodyfixed id="Bottom" mkbound="0" modelfile="AutoActual" modelnormal="invert" />
    <bodyfloating id="Floaters" mkbound="51-52" modelfile="AutoActual"/>
    <bodyfloating id="FloaterLink" mkbound="50" modelfile="AutoActual"/>
    <link_linearspring idbody1="Bottom" idbody2="FloaterLink">
        <point_fb1 x="1.15" y="0.0" z="0.0" comment="Point in body 1" />
        <point_fb2 x="1.15" y="0.0" z="0.325" comment="Point in body 2" />
        <stiffness value="100.0" comment="Stiffness [N/m]" />
        <damping value="500.0" comment="Damping [Ns/m]" />
        <rest_length value="0.325" comment="Spring equilibrium length [m]" />
        <savevtk>
            <nside value="16" comment="number of sections for each revolution. (default=16)" />
            <radius value="5.0" comment="spring radius (default=3)" />
            <length value="3.0" comment="length for each revolution (default=1)" />
        </savevtk>
    </link_linearspring>
</chrono>
```

Figura 26. Configuración del caso de validación de la salida de datos

Se ejecuta la simulación y se generan 5 ficheros de salida CSV, tal como se puede observar en la Figura 27.

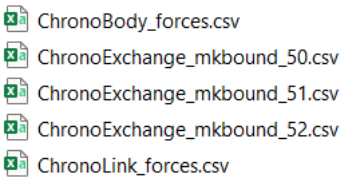

Figura 27. Ficheros CSV generados en el caso de validación de la salida de datos

La salida de datos mediante ficheros CSV se realiza correctamente, por ello esta iteración queda validada e integrada.

8.4 ITERACIÓN 6. MÓDULO PARALELO

En esta iteración se valida la implementación de simulaciones con el módulo *PARALLEL* de Chrono para reducir el tiempo de ejecución en Chrono.

Se define un caso con un objeto fijo *(Domain*) y 28 objetos flotantes: 27 cajas (*Boxes*) y una esfera (*Ball*). En este caso existe un gran número de objetos colisionando entre sí, por ello es un caso óptimo para validar el módulo paralelo y ver la aceleración con respecto a la ejecución secuencial. Se activa el módulo paralelo con la opción *ompthreads="0"* (crea tantos *threads* como número de núcleos tenga el procesador). La Figura 28 muestra la estructura del fichero de configuración XML.

```
<chrono>
    <savedata value="0.01" comment="Saves CSV with data exchange for each time interval (0=all steps)" />
    <collision activate="true">
        <distancedp value="0.5" comment="Allowed collision overlap according Dp (default=0.5)" />
        <ompthreads value="0" comment="Number of threads by host for parallel execution.
                                       0:Multi-Core, 1:Single-Core (default=1)" />
    </collision>
    <bodyfixed id="Domain" mkbound="0" modelfile="AutoActual" modelnormal="invert" />
    <bodyfloating id="Ball" mkbound="10" modelfile="AutoActual" />
    <bodyfloating id="Boxes" mkbound="51-77" modelfile="AutoActual" />
</chrono>
```

Figura 28. Configuración del caso de validación del módulo paralelo

La Figura 29 muestra la salida por consola de la ejecución de este caso. Se puede observar que se está ejecutando en modo *Multi-Core* con 8 *threads* y con las colisiones activadas.

```
C:\WINDOWS\system32\cmd.exe

Chrono Objects configuration:
Executing Chrono_v4.0.0.001
  Data directory....: [Test_parallel_out/chrono_objs/]
  Collisions........: True
    Collision dp....: 0.5
    Collision shapes: 749
  Execution mode....: Multi Core
  OpenMP Threads....: 8
  Bodies............: 29
  Links.............: 0
  Body_0000 "domain" -  type: Fixed
    MkBound......: 0
    Kfric........: 0.45
    Restitution..: 0.8
    ModelFile....: domain_mkb0000.obj
    ModelNormal..: Invert
  Body_0001 "ball" -  type: Floating
    MkBound......: 10
    Mass.........: 1186.8
    Center.......: (7.5,1.86077e-16,0.26)
    Inertia......: (21.2049,21.2049,21.2049) (xx,yy,zz)
    Inertia......: (-7.40241e-16,1.65389e-16,1.2948e-16) (xy,yz,xz)
    Kfric........: 0.45
    Restitution..: 0.8
    ModelFile....: ball_mkb0010.obj
    ModelNormal..: Original
  Body_0002 "box51" -  type: Floating
    MkBound......: 51
    Mass.........: 30.016
```

Figura 29. Salida por consola del caso de validación del módulo paralelo

El resultado de la simulación es el esperado (Figura 30). Tanto el módulo paralelizado como las colisiones funcionan correctamente.

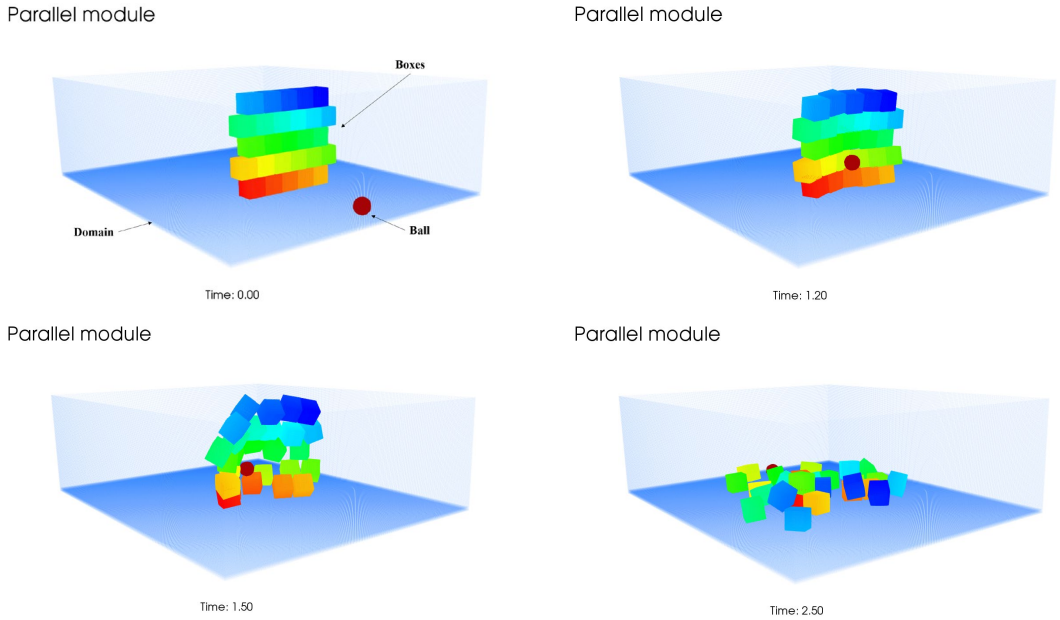

Figura 30. Secuencia de simulación del caso de validación del módulo paralelo

En la Tabla 5 se muestran los tiempos de ejecución de la librería de Chrono y del total de simulación entre DualSPHysics y Chrono. Se puede apreciar que hay una reducción del 30% en el tiempo de ejecución de Chrono usando el módulo paralelo (con 8 hilos de ejecución) con respecto al código secuencial y una aceleración de 1.41x.

Simulación	Multi-Core	Single-Core
Chrono	250.02 s	353.79 s
Total	769.21 s	885.54 s

Tabla 6. Tiempos de simulación del caso de validación del módulo paralelo

El módulo paralelo funciona correctamente y reduce el tiempo de simulación con respecto al código secuencial, por ello esta iteración queda validada e integrada en el sistema.

8.5 ITERACIÓN 7. COLISIONES

En esta iteración se realizan las pruebas para validar las colisiones elásticas (*SMooth Contacts,* SMC) y se compara su comportamiento con las colisiones rígidas existentes anteriormente (*Non-Smooth Contacts*, NSC).

Se define un caso con el método de colisión SMC (*contactmethod="1"*) con un objeto flotante (*Box*) al que se le aplica una velocidad inicial y un objeto rígido (*Domain*) con el que colisionará el objeto flotante. La Figura 31 muestra la estructura del fichero de configuración XML. También se define otro caso con las mismas características, pero utilizando el tipo de colisión rígida (*Non-Smooth Contacts*, NSC) para analizar la diferencia entre ambos métodos.

```
<chrono>
    <savedata value="0.01" comment="Saves CSV with data exchange for each time interval (0=all steps)" />
    <collision activate="true">
        <distancedp value="0.5" comment="Allowed collision overlap according Dp (default=0.5)" />
        <contactmethod value="1" comment="Contact method type.
                                          0:NSC (Non Smooth Contacts), 1:SMC (SMooth Contacts).(default=0)" />
    </collision>
    <bodyfixed id="Domain" mkbound="10" modelfile="AutoActual"/>
    <bodyfloating id="Box" mkbound="50" modelfile="AutoActual"/>
</chrono>
```

Figura 31. Configuración del caso de validación de las colisiones SMC

Se realiza una simulación de 3 segundos de ambos casos, cuya secuencia se puede observar en la Figura 32.

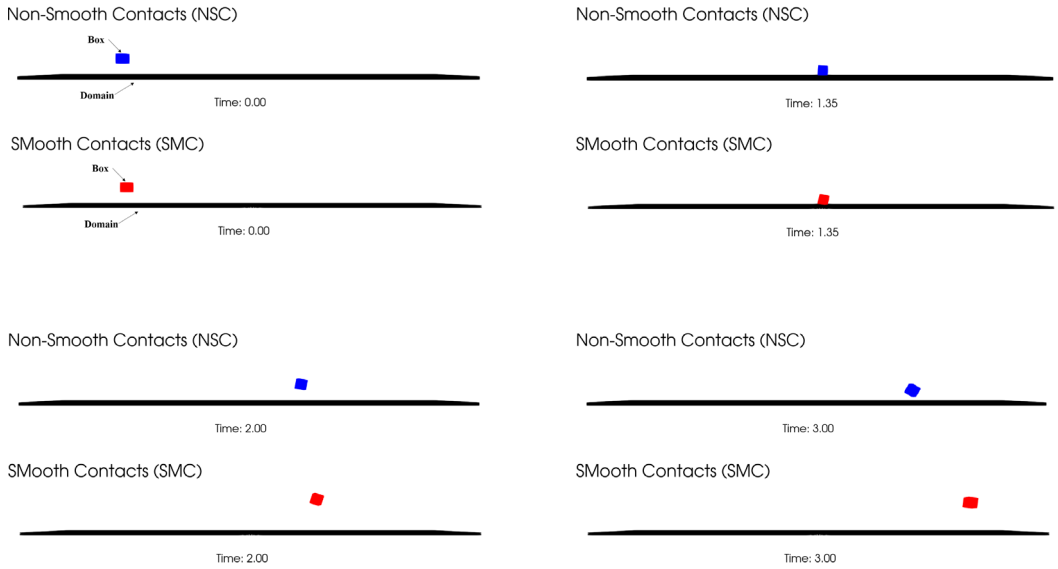

Figura 32. Secuencia de simulación para la validación de las colisiones SMC

El resultado muestra que las colisiones SMC presentan una mayor elasticidad y por ello se elevan más a consecuencia de la colisión con el suelo. Este comportamiento se puede observar en el gráfico de la Figura 33, analizando la posición del centro de masas del objeto en el eje Z (*center.z*) a lo largo de la simulación.

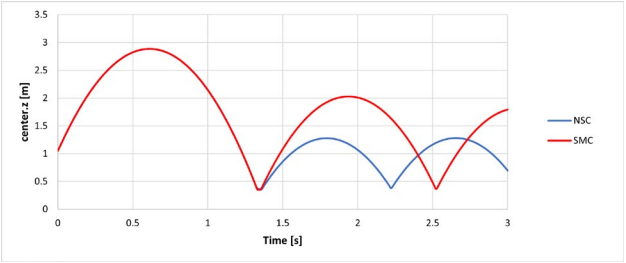

Figura 33. Centro de masas en Z del objeto flotante con dos métodos de colisión

En esta iteración también se prueba la imposición de coeficientes de fricción. Para ello, se reutiliza el caso anterior con colisiones NSC y se define la imposición del coeficiente de fricción del objeto flotante añadiendo el atributo *imposefric*="*true*". El resultado de la ejecución se puede observar en la Figura 34, donde se aprecia que varía el comportamiento del objeto al producirse la colisión y rozar con el suelo.

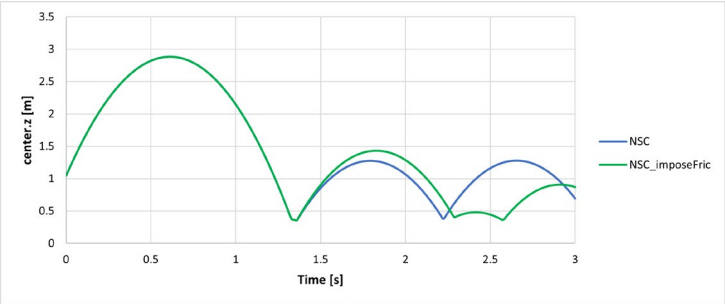

Figura 34. Gráfica de la posición en Z del objeto flotante con colisiones NSC

Todas las características implementadas funcionan correctamente, por lo tanto, esta iteración queda validada e integrada en el sistema.

8.6 ITERACIÓN 8. ENLACES ENTRE OBJETOS

En esta iteración se prueban los nuevos *links* implementados y la funcionalidad de variar los coeficientes de *stiffness* y *damping* durante las simulaciones.

Se define un caso usando el *link Coulomb damping* que conecta a un objeto flotante (*Floater*) con un objeto rígido (*Bottom*). También se define el fluido y un pistón para la generación de oleaje en DualSPHysics. La Figura 35 muestra la estructura del fichero de configuración XML para definir este caso.

```
<chrono>
    <savedata value="0.05" comment="Saves CSV with data exchange for each time interval (0=all steps)" />
    <bodyfixed id="Bottom" mkbound="0" />
    <bodyfloating id="Floater" mkbound="50" />
    <link_coulombdamping  idbody1="Bottom" idbody2="Floater">
        <point_fb1 x="2.0" y="0.0" z="0.0" comment="Point in body 1" />
        <point_fb2 x="2.0" y="0.0" z="0.325" comment="Point in body 2" />
        <rest_length value="0.325" comment="Spring equilibrium length [m]" />
        <damping value="10.0" comment="Coulomb force [N]" />
        <savevtk>
            <nside value="16" comment="number of sections for each revolution.(default=16)" />
            <radius value="5.0" comment="spring radius (default=3)" />
            <length value="3.0" comment="length for each revolution (default=1)" />
        </savevtk>
    </link_coulombdamping>
</chrono>
```

Figura 35. Configuración del caso de validación del link Coulomb damping

La Figura 36 muestra distintos instantes de la simulación. Se puede observar que está funcionando correctamente ya que el objeto flotante está anclado al muelle y solamente se mueve verticalmente al interactuar con el oleaje.

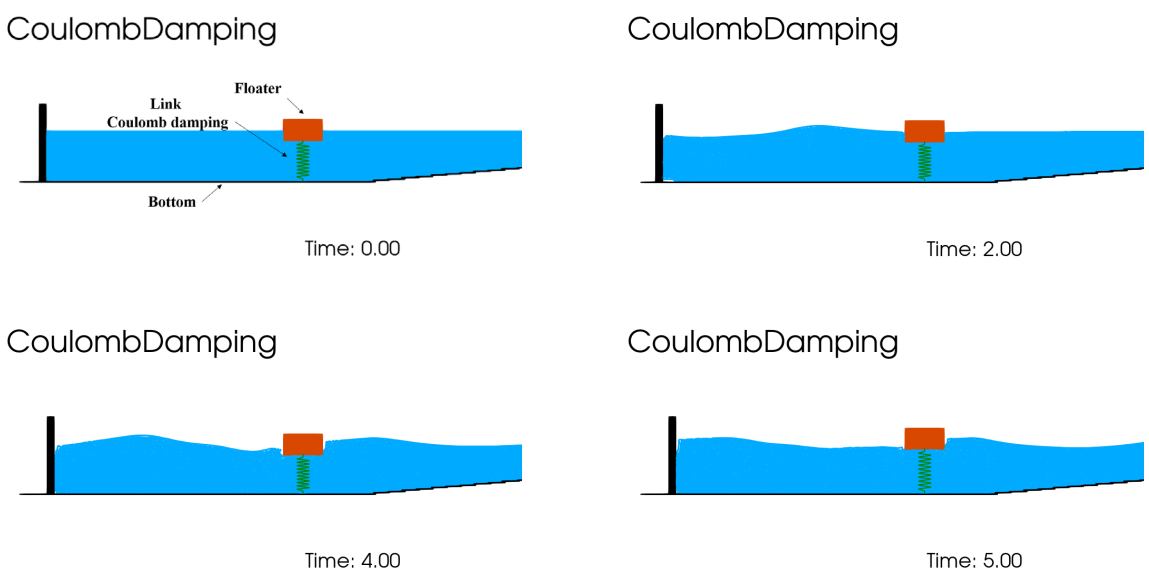

Figura 36. Secuencia de simulación del link Coulomb damping

Se define un caso usando el *link Pulley* con dos objetos flotantes (*Wheel* y *Gear*), un objeto fijo (*Domain*), una bisagra (*link_hinge*) que permitirá el giro del *Wheel* y una polea (*link_pulley*) que conecta al objeto *Wheel* (maestro) con el *Gear* (esclavo). También se define el fluido que estará en movimiento e impondrá el giro del *Wheel*. La Figura 37 muestra la estructura del fichero de configuración XML.

```
<chrono>
    <savedata value="0.01" comment="Saves CSV with data exchange for each time interval (0=all steps)" />
    <bodyfixed id="Domain" mkbound="0" modelfile="AutoActual"/>
    <bodyfloating id="Wheel" mkbound="50" modelfile="AutoActual"/>
    <bodyfloating id="Gear" mkbound="51" modelfile="AutoActual"/>
    <link_hinge idbody1="Wheel">
        <rotpoint x="1.0" y="0" z="0.6" comment="Point for rotation" />
        <rotvector x="0.0" y="1.0" z="0.0" comment="Vector direction for rotation" />
        <stiffness value="0.0" comment="Torsional stiffness [Nm/rad]" />
        <damping value="0.0" comment="Torsional damping [Nms/rad]" />
    </link_hinge>
    <link_pulley idbody1="Wheel" idbody2="Gear">
        <rotpoint x="1.0025" y="0" z="1.2025" comment="Point for rotation" />
        <rotvector x="0.0" y="1.0" z="0.0" comment="Rotation axis of the body" />
        <radius value="0.3" comment="Radius of idbody1"/>
        <radius2 value="0.05" comment="Radius of idbody2"/>
    </link_pulley>
</chrono>
```

Figura 37. Configuración del caso de validación del link pulley

Se realiza la simulación y el objeto *Wheel* gira al verse influenciado por el movimiento del fluido. El movimiento del *Wheel* a su vez impone el giro del *Gear*. En la Figura 38 se puede observar este comportamiento.

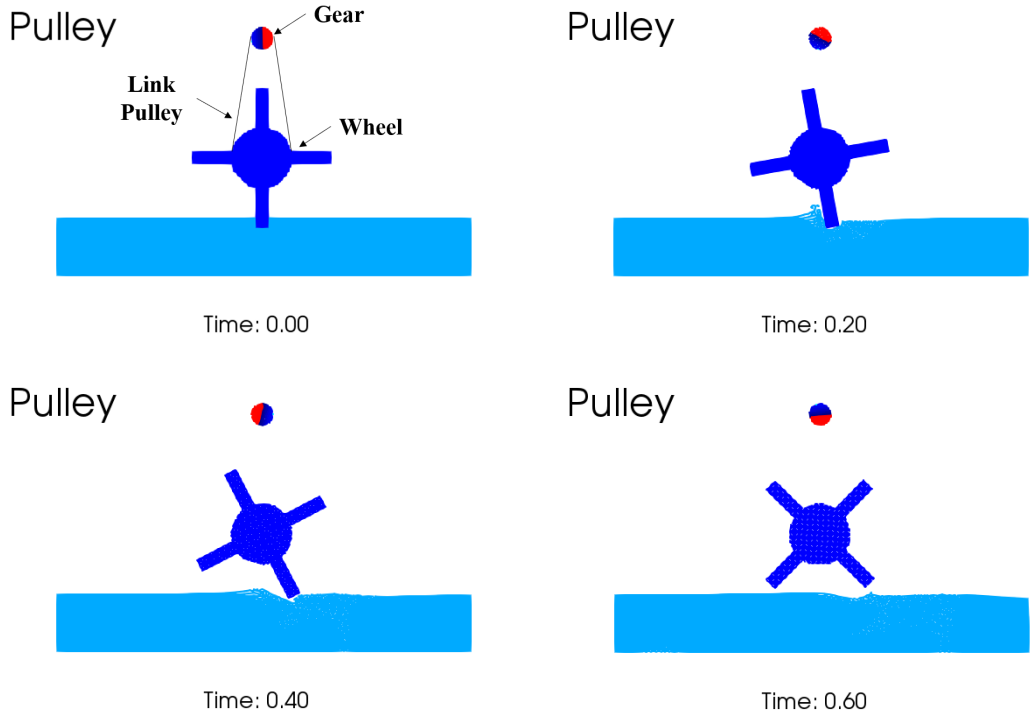

Figura 38. Secuencia de simulación del link pulley

La última prueba de esta iteración sirve para validar la funcionalidad de variar los coeficientes de *stiffness* (*k*) y *damping* (*c*). Para ello, se reutiliza la configuración del *linear spring* del caso mostrado en la Figura 24, cuyos coeficientes son *k=100 [N/m]* y *c=500 [Ns/m]*. Se realizan dos simulaciones: una con los coeficientes fijos y otra con coeficientes variables.

La simulación con coeficientes variables comienza con coeficientes nulos *k=c=0* y a partir del segundo 5, se aplican los coeficientes *k=100 [N/m]* y *c=500 [Ns/m]*. El resultado de la simulación se puede observar en la Figura 39, donde se comparan las elevaciones de los objetos de ambas simulaciones. A partir del segundo 5 ambos tienen los mismos valores de coeficientes, por ello las elevaciones coinciden.

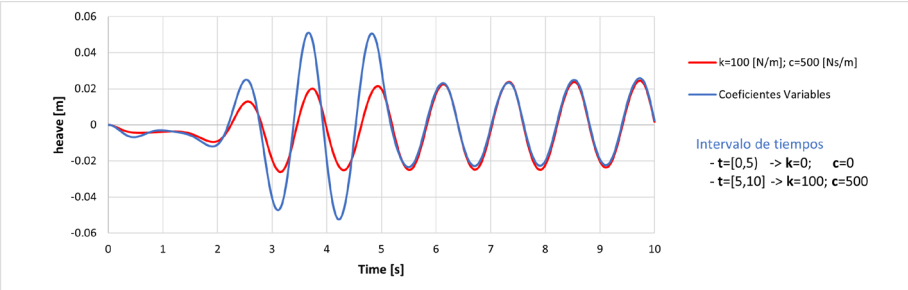

Figura 39. Elevación en Z del objeto flotante conectado a un muelle lineal

Todas las funcionalidades incorporadas en esta iteración funcionan correctamente, por lo tanto, quedan validadas e integradas en el sistema.

8.7 PRUEBAS GLOBALES

En este apartado se documentan las pruebas globales que sirven para validar el sistema. Las pruebas consisten en obtener datos de la simulación de un caso y compararlos con datos experimentales. Las dimensiones del caso de validación se pueden observar en la Figura 40.

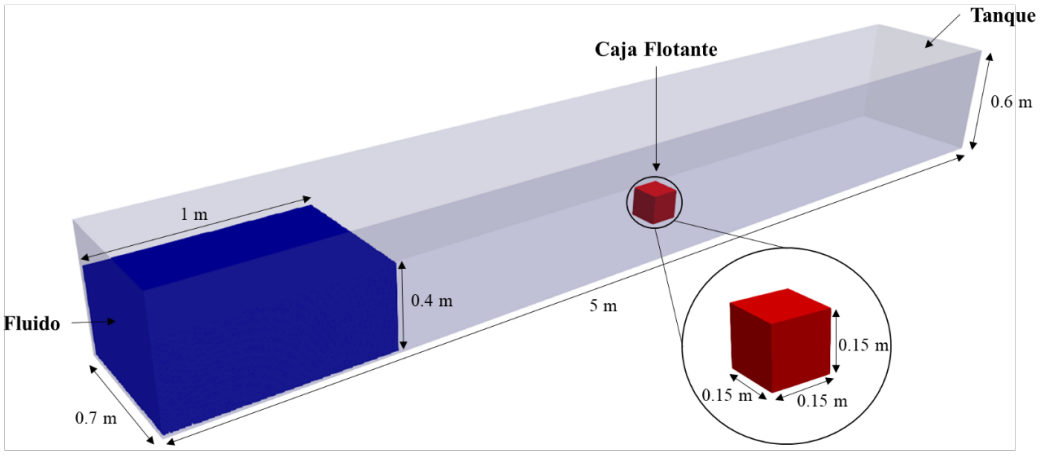

Figura 40. Dimensiones del dominio del caso de validación global

El caso está formado por un objeto flotante (*Cube*), un objeto fijo (*Tank*) y un fluido que desplazará al objeto flotante en el eje X. La estructura del fichero de configuración XML se puede observar en la Figura 41.

```
<chrono>
    <savedata value="0.01" comment="Saves CSV with data exchange for each time interval (0=all steps)" />
    <collision activate="true">
        <distancedp value="0.5" comment="Allowed collision overlap according Dp (default=0.5)" />
        <contactmethod value="0" comment="Contact method type.
                                          0:NSC (Non Smooth Contacts), 1:SMC (SMooth Contacts). (default=0)" />
    </collision>
    <bodyfloating id="Cube" mkbound="51" modelfile="AutoActual" />
    <bodyfixed id="Tank" mkbound="0" modelfile="AutoActual" modelnormal="invert" />
</chrono>
```

Figura 41. Configuración del caso de validación global

Se realiza la simulación y se obtienen los datos del desplazamiento del objeto. La Figura 42 muestra la secuencia de simulación, donde se aprecia el movimiento del objeto flotante provocado por la interacción con el fluido.

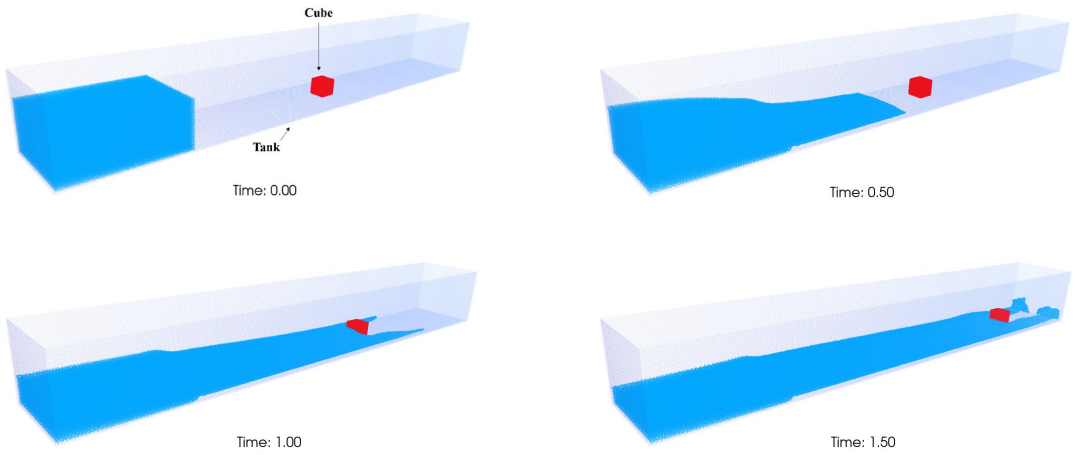

Figura 42. Secuencia de la simulación del caso de validación global

Analizando los datos, se puede observar en la Figura 43 que el resultado de la simulación es muy próximo al obtenido de forma experimental.

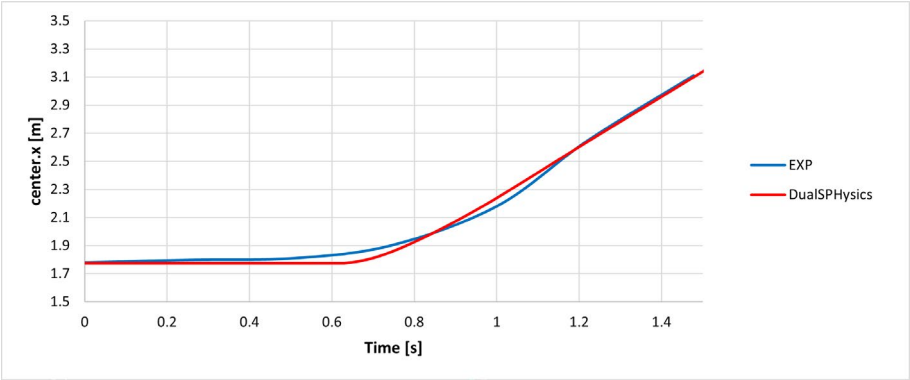

Figura 43. Valores de desplazamiento experimentales (EXP) y numéricos (DualSPHysics)

Tras analizar los datos obtenidos, se garantiza que el acoplamiento funciona correctamente y que obtiene unos resultados numéricos muy próximos a los resultados experimentales, el sistema se da por validado y se cumple el objetivo principal de este trabajo.

9 MANUAL DE USUARIO

Este manual está orientado a explicar cómo utilizar el acoplamiento entre DualSPHysics y Chrono. Es necesario disponer del directorio **DualSPHysicsChrono** que se suministra en el CD. En el ANEXO 2. ESTRUCTURA DE DUALSPHYSICSCHRONO se muestra la estructura de ficheros y directorios contenidos en DualSPHysicsChrono.

En este manual también se documentan los procesos de compilación, a pesar de que el proyecto adjunto con este trabajo ya contiene todos los ejecutables y librerías necesarias. Al tratarse de código libre, la idea es que cualquier usuario pueda realizar cambios y posteriormente compilar el código fuente.

9.1 COMPILACIÓN

En este apartado se documenta el proceso de compilación y los aspectos a tener en cuenta para generar las librerías y ejecutables necesarios para utilizar el acoplamiento.

9.1.1 CHRONO

El código de este proyecto está implementado para usar Chrono-4.0.0 y se puede descargar del repositorio oficial de ProjectChrono (https://github.com/projectchrono/chrono). Algunos ficheros de la versión original fueron modificados para añadir nuevas funcionalidades. Estos ficheros se encuentran en el directorio **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Source/ChronoChanges**. Será necesario copiar el contenido de esa carpeta y reemplazar los ficheros dentro del directorio **chrono-4.0.0**. El proceso de compilación adaptado al acoplamiento con DualSPHysics se puede consultar en el ANEXO 3. COMPILACIÓN DE CHRONO.

9.1.2 DSPHCHRONOLIB

La librería DSPChronoLib permite la comunicación entre DualSPHysics y Chrono. Para poder compilarla, es necesario haber compilado previamente el proyecto de Chrono y haber generado las librerías dinámicas de Chrono. En caso de utilizar el módulo de computación paralela de Chrono, es necesario descargar la librería Blaze versión 3.2 (https://bitbucket.org/blaze-lib/blaze/downloads/).

La construcción y compilación de la librería usando CMake (https://cmake.org/) necesita las siguientes dependencias:

- CMake versión 3.0.0 o superior.
- Microsoft Visual C++ 2015 (v140) en plataformas Windows.
- GNU G++ 4.9 o superior en plataformas Linux.
- Que exista el fichero "CMakeLists.txt" en **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Source**.

A continuación, se muestran los pasos para la compilación:

1. Abrir CMake (cmake-gui).
2. Introducir la ruta **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Source** en el campo "Where is the source code".
3. Introducir la ruta **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Build** en el campo "Where to build the binaries".

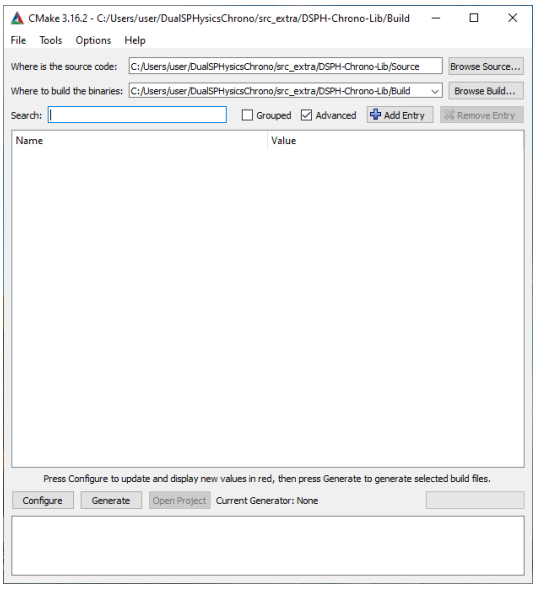

Figura 44. Introducir las rutas para la generación de DSPHChronoLib

4. Pulsar en “Configure”.
5. Seleccionar “Visual Studio 14 2015” y la plataforma “x64” para Windows (Figura 45) y “Unix Makefiles” para Linux (Figura 46).

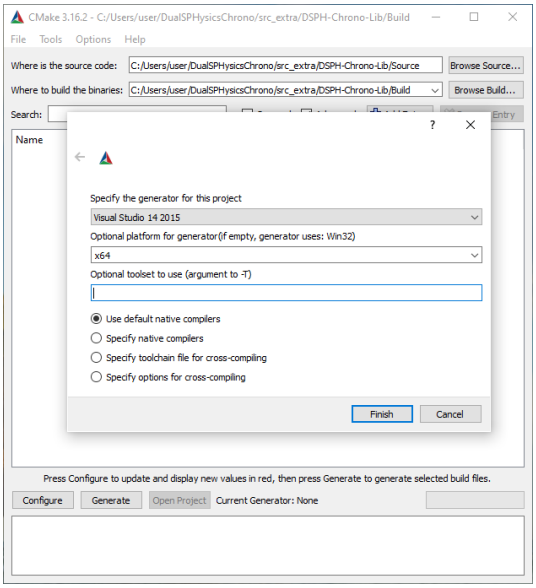

Figura 45. Selección de compiladores en Windows para DSPHChronoLib

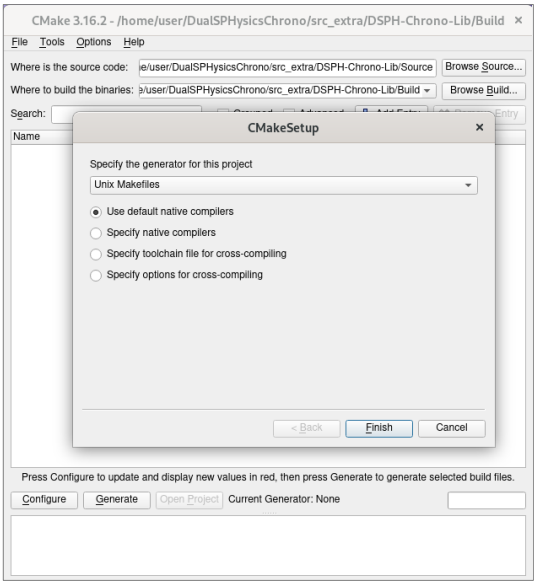

Figura 46. Selección de compiladores en Linux para DSPHChronoLib

6. Introducir la ruta del directorio **chrono-4.0.0_build/cmake** en el campo ***Chrono_DIR***.

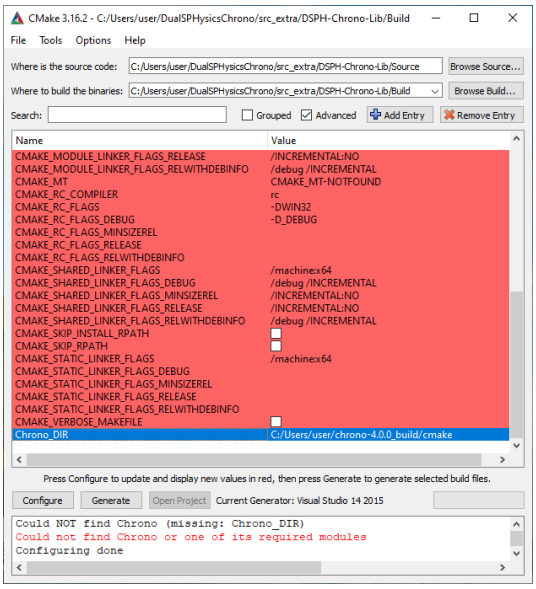

Figura 47. Introducir la ruta de la librería Chrono

7. Pulsar en “Configure”.
8. En caso de utilizar el módulo paralelo, se deberá introducir la ruta de la librería Blaze en el campo “BLAZE_DIR”.

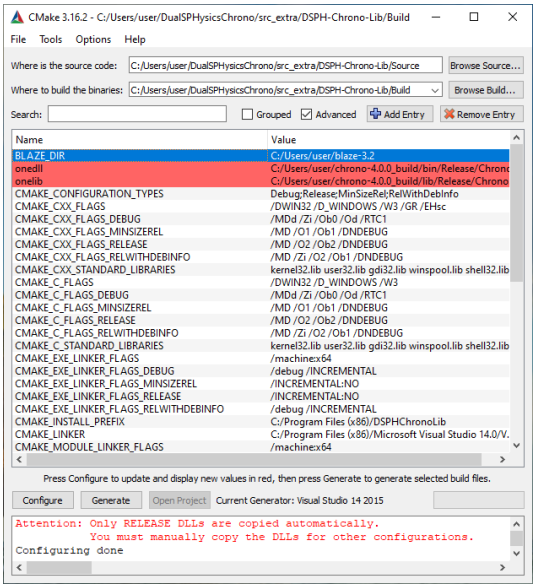

Figura 48. Introducir la ruta de la librería Blaze

9. Pulsar en “Configure”.
10. Pulsar en “Generate” y en la ventana inferior de CMake deberá mostrarse “Generating done”.

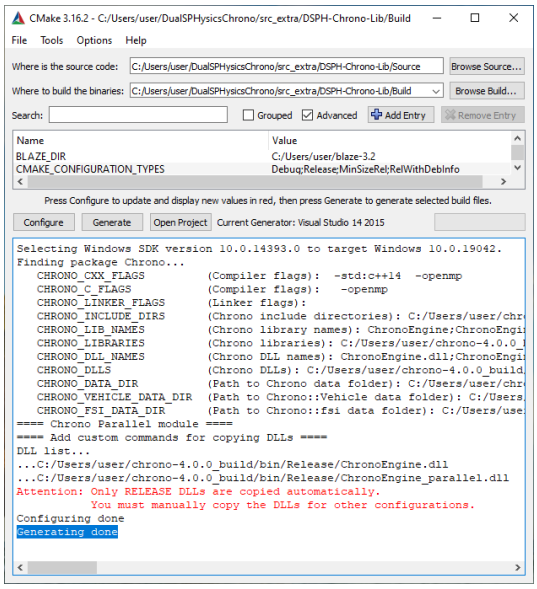

Figura 49. Generación de la solución de DSPHChronoLib

WINDOWS

Si los pasos anteriores se realizaron correctamente, se generará una solución denominada **DSPHChronoLib.sln** en el directorio **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Build**.

11.1. Abrir la solución con Microsoft Visual Studio 2015.

12.1. Seleccionar modo de compilación “Release”.

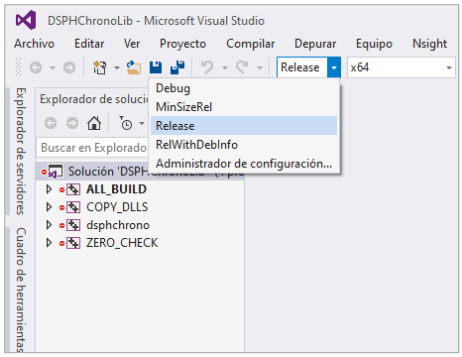

Figura 50. Selección del modo de compilación de DSPHChronoLib

13.1. Pulsar en “Compilar” y a continuación, en “Compilar solución”.

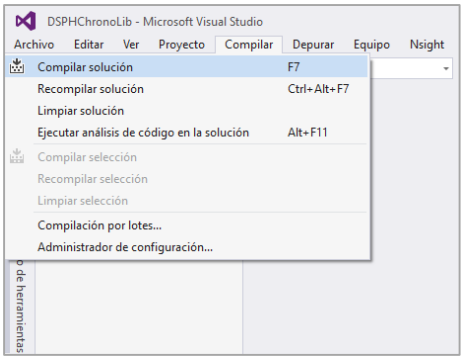

Figura 51.Compilar DSPHChronoLib con Microsoft Visual Studio 2015

Al finalizar la compilación se generarán los ficheros **dsphchrono.dll** y **dsphchrono.lib** en **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Build/Release**. Copiar el fichero **dsphchrono.dll** en **DualSPHysicsChrono/bin/windows** y el fichero **dsphchrono.lib** en **DualSPHysicsChrono/src/lib/vs2015**. Copiar los ficheros **DSPHChronoLib.h** y **JChronoData.h** contenidos en **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Source** a **DualSPHysicsChrono/src/source.**

LINUX

Si los pasos anteriores se realizaron correctamente, se generará un fichero *Makefile* en **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Build**.

11.2. Abrir un terminal de Linux en el directorio **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Build**.

12.2. Compilar el código usando Make.

Figura 52. Compilar DSPHChronoLib con Make en Linux

Al finalizar la compilación, se generará un fichero **libdsphchrono.so** en **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Build**. Copiar este fichero en **DualSPHysicsChrono/src/lib/linux_gcc**. Copiar los ficheros **DSPHChronoLib.h** y **JChronoData.h** contenidos en **DualSPHysicsChrono/src_extra/DSPH-Chrono-Lib/Source** a **DualSPHysicsChrono/src/source.**

9.1.3 DUALSPHYSICS

DualSPHysics ofrece tanto una solución de Microsoft Visual Studio 2015, como el fichero *Makefile* para poder compilar el código con las dependencias sin necesidad de generar los proyectos. El proceso de compilación se documenta en el ANEXO 4. COMPILACIÓN DE DUALSPHYSICS.

9.2 EJECUCIÓN

Se incorporan dos ejecutables para Windows y otros dos ejecutables para Linux en el directorio **DualSPHysicsChrono/bin** para poder lanzar los casos del paquete suministrado. Uno de ellos ejecuta DualSPHysics en CPU y el otro puede ejecutar DualSPHysics tanto en CPU como en GPU. En cada carpeta de ejemplos del directorio (**DualSPHysicsChrono/examples**), se incorporan varios scripts que permiten lanzar los ejecutables del directorio **bin**. También se incluyen los ficheros necesarios para la simulación de cada ejemplo. En este apartado se explica cómo se puede lanzar una simulación de DualSPHysics con Chrono, tomando como ejemplo el caso **DualSPHysicsChrono/examples/chrono/00_CaseManual**. El contenido de este directorio es el siguiente:

- **CaseManual_Def.xml**: Fichero de configuración de DualSPHysics y Chrono.
- **CaseManual_Thumbail.mp4**: Vídeo reducido con el resultado de la ejecución.
- **Command Prompt**: Símbolo del sistema utilizado en Windows para ejecutar comandos.
- **wCaseManual_win64_CPU.bat**: Script para ejecutar el caso en CPU en Windows.
- **wCaseManual_win64_GPU.bat**: Script para ejecutar el caso en CPU&GPU en Windows.
- **xCaseManual_linux64_CPU.sh**: Script para ejecutar el caso en CPU en Linux.
- **xCaseManual_linux64_GPU.sh**: Script para ejecutar el caso en CPU&GPU en Linux.

WINDOWS

1. Abrir **Command Prompt** en el directorio del ejemplo a ejecutar.

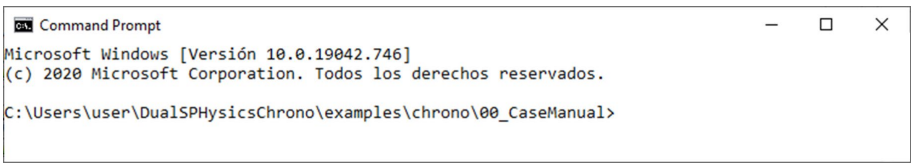

Figura 53. Command prompt de Windows

2. Ejecutar el script con extensión *.bat* que se desee (si no se dispone de una GPU compatible con CUDA o no se instalaron los drivers de Cuda 9.2 o posterior, ejecutar el script de CPU).

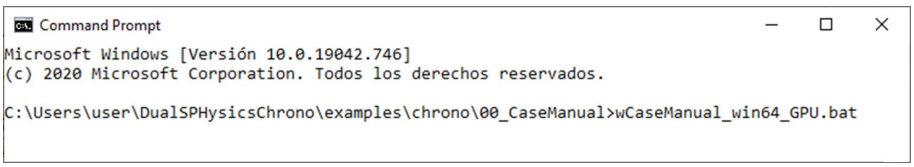

Figura 54. Ejecución de una simulación de DualSPHysics en Windows

LINUX

1. Abrir un **Terminal** de Linux y acceder al directorio **DualSPHysicsChrono**.

Figura 55. Abrir un terminal de Linux en DualSPHysicsChrono

2. Dar permisos de ejecución al script **chpermissions.sh** y ejecutarlo.

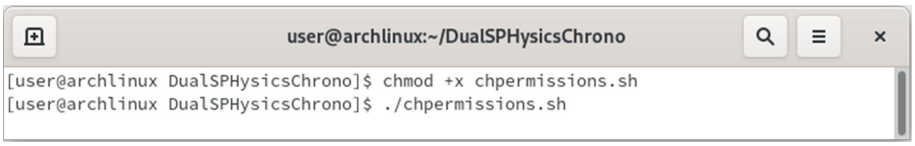

Figura 56. Dar permisos de ejecución a scripts y ejecutables de Linux

3. Acceder al directorio del ejemplo a ejecutar, en este caso **00_CaseManual**.

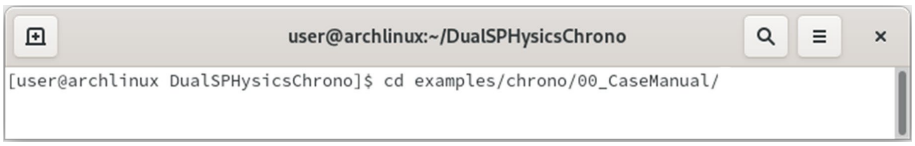

Figura 57. Acceder al directorio 00_CaseManual

4. Ejecutar el script con extensión *.sh* que se desee (si no se dispone de una GPU compatible con CUDA o no se instalaron los drivers de Cuda 9.2 o posterior, ejecutar el script de CPU).

Figura 58. Ejecución de una simulación de DualSPHysics en Linux

CONCLUSIONES

En este trabajo se ha desarrollado un nuevo acoplamiento entre DualSPHysics y Chrono y se han añadido nuevas funcionalidades con respecto a la versión que existía anteriormente. Estas nuevas funcionalidades dotan a DualSPHysics de más características, ampliando la variedad de simulaciones que se pueden llevar a cabo con este modelo. Destacando especialmente, la incorporación de una nueva versión paralelizada que permite reducir significativamente los tiempos de ejecución de las simulaciones con respecto al código secuencial.

En el desarrollo de este trabajo se ha seguido una estrategia que facilita el acoplamiento de DualSPHysics con otros modelos o con nuevas versiones de Chrono. El uso de la librería DSPHChronoLib mantiene una alta cohesión y a la vez un bajo acoplamiento entre ambos modelos, ya que actúa como interfaz de comunicación y traduce las peticiones/respuestas entre ambos modelos. De esta forma, se garantiza el correcto funcionamiento de DualSPHysics manteniendo siempre el mismo flujo de comunicación, independientemente del modelo o la versión de Chrono acoplada.

Este acoplamiento entre DualSPHysics y Chrono se han validado comparando los resultados numéricos con datos experimentales, garantizando que los resultados son precisos y, por consiguiente, ambos modelos funcionan correctamente de forma conjunta.

Teniendo en cuenta los buenos resultados obtenidos y el conjunto de funcionalidades existentes en este acoplamiento, se valora la posibilidad de que el presente trabajo dé lugar a alguna publicación científica. Además, este código se liberará de forma gratuita para que sea accesible a toda la comunidad de DualSPHysics y Chrono.

REFERENCIAS

[1] A. J. Crespo *et al.*, «DualSPHysics: Open-source parallel CFD solver based on Smoothed Particle Hydrodynamics (SPH)», *Comput. Phys. Commun.*, vol. 187, pp. 204-216, 2015.

[2] A. C. Crespo, J. M. Dominguez, A. Barreiro, M. Gómez-Gesteira, y B. D. Rogers, «GPUs, a new tool of acceleration in CFD: efficiency and reliability on smoothed particle hydrodynamics methods», *PloS One*, vol. 6, n.º 6, p. e20685, 2011.

[3] C. Altomare, A. J. Crespo, J. M. Domínguez, M. Gómez-Gesteira, T. Suzuki, y T. Verwaest, «Applicability of smoothed particle hydrodynamics for estimation of sea wave impact on coastal structures», *Coast. Eng.*, vol. 96, pp. 1-12, 2015.

[4] C. Altomare *et al.*, «Long-crested wave generation and absorption for SPH-based DualSPHysics model», *Coast. Eng.*, vol. 127, pp. 37-54, 2017.

[5] M. Gomez-Gesteira, B. D. Rogers, A. J. Crespo, R. A. Dalrymple, M. Narayanaswamy, y J. M. Dominguez, «SPHysics–development of a free-surface fluid solver–Part 1: Theory and formulations», *Comput. Geosci.*, vol. 48, pp. 289-299, 2012.

[6] M. Gómez-Gesteira, A. J. Crespo, B. D. Rogers, R. A. Dalrymple, J. M. Dominguez, y A. Barreiro, «SPHysics–development of a free-surface fluid solver–Part 2: Efficiency and test cases», *Comput. Geosci.*, vol. 48, pp. 300-307, 2012.

[7] J. M. Domínguez, A. J. Crespo, y M. Gómez-Gesteira, «Optimization strategies for CPU and GPU implementations of a smoothed particle hydrodynamics method», *Comput. Phys. Commun.*, vol. 184, n.º 3, pp. 617-627, 2013.

[8] A. Tasora *et al.*, «Chrono: An open source multi-physics dynamics engine», en *International Conference on High Performance Computing in Science and Engineering*, 2015, pp. 19-49.

[9] M. Brito *et al.*, «Coupling between DualSPHysics and Chrono-Engine: towards large scale HPC multiphysics simulations», 2016.

[10] I. Sommerville, *Software engineering*. Pearson Education India, 2004.

[11] I. Jacobson, G. Booch, y J. Rumbaugh, *UML: el proceso unificado de desarrollo de software*. Addison-Wesley, 2000.

[12] B. Stroustrup, *The C++ programming language*. Pearson Education, 2013.

[13] B. J. Gough y R. Stallman, *An Introduction to GCC.* Citeseer, 2004.

[14] R. Stallman, R. McGrath, y P. Smith, «GNU make», *Free Softw. Found. Boston*, 1988.

[15] R. Stallman, R. Pesch, y S. Shebs, «Debugging with GDB», *Free Softw. Found.*, vol. 675, 1988.

[16] K. Martin y B. Hoffman, *Mastering CMake: a cross-platform build system: version 3.1*. Kitware, 2015.

[17] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, y F. Yergeau, *Extensible markup language (XML) 1.0*. W3C recommendation October, 2000.

[18] «User's Guide», *Visual Paradigm Community Circle*. https://circle.visual-paradigm.com/docs/ (accedido ene. 11, 2021).

[19] P. Ritchie, «Introduction to Visual Studio 2015», en *Practical Microsoft Visual Studio 2015*, Springer, 2016, pp. 1-25.

ANEXOS

ANEXO 1. FLUJO DE TRABAJO DE DUALSPHYSICS

El flujo de ejecución de DualSPHysics se divide en tres bloques: *pre-processing*, *processing* y *post-processing*. La Figura 59 muestra las herramientas y los ficheros utilizados en todo el proceso de simulación.

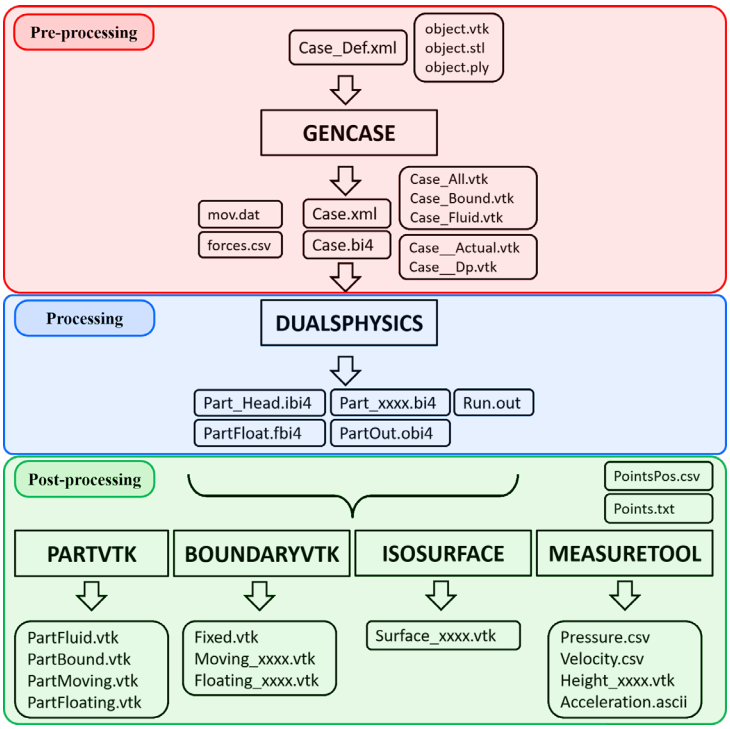

Figura 59. Flujo de trabajo de DualSPHysics
Fuente: https://dual.sphysics.org/

PRE-PROCESSING

La herramienta GenCase define el estado inicial de la simulación y los parámetros de ejecución de DualSPHysics. Utiliza un fichero de configuración de entrada XML (*Case_Def.xml*) y geometrías externas, a partir de los cuales, discretiza la geometría del dominio en un conjunto de puntos que se corresponderán con las posiciones de las partículas.

La ejecución de GenCase genera dos ficheros de salida:

- *Case.xml*. Es una copia del fichero *Case_Def.xml*, al que se le añade información sobre las partículas creadas para representar el dominio.
- *Case.bi4*. Contiene el estado inicial del dominio almacenando el número de partículas, sus posiciones, velocidades y densidades.

PROCESSING

En este bloque se realiza la simulación mediante el método SPH. DualSPHysics es el modelo encargado de leer los ficheros de salida del bloque de *pre-processing* y ejecutar la simulación teniendo en cuenta los parámetros de ejecución definidos en el fichero *Case_Def.xml*.

POST-PROCESSING

En el bloque de *post-processing* existen varias herramientas que extraen información de los ficheros binarios (extensión *.bi4*) generados por DualSPHysics para analizar o visualizar la información resultante del proceso de simulación.

ANEXO 2. ESTRUCTURA DE DUALSPHYSICSCHRONO

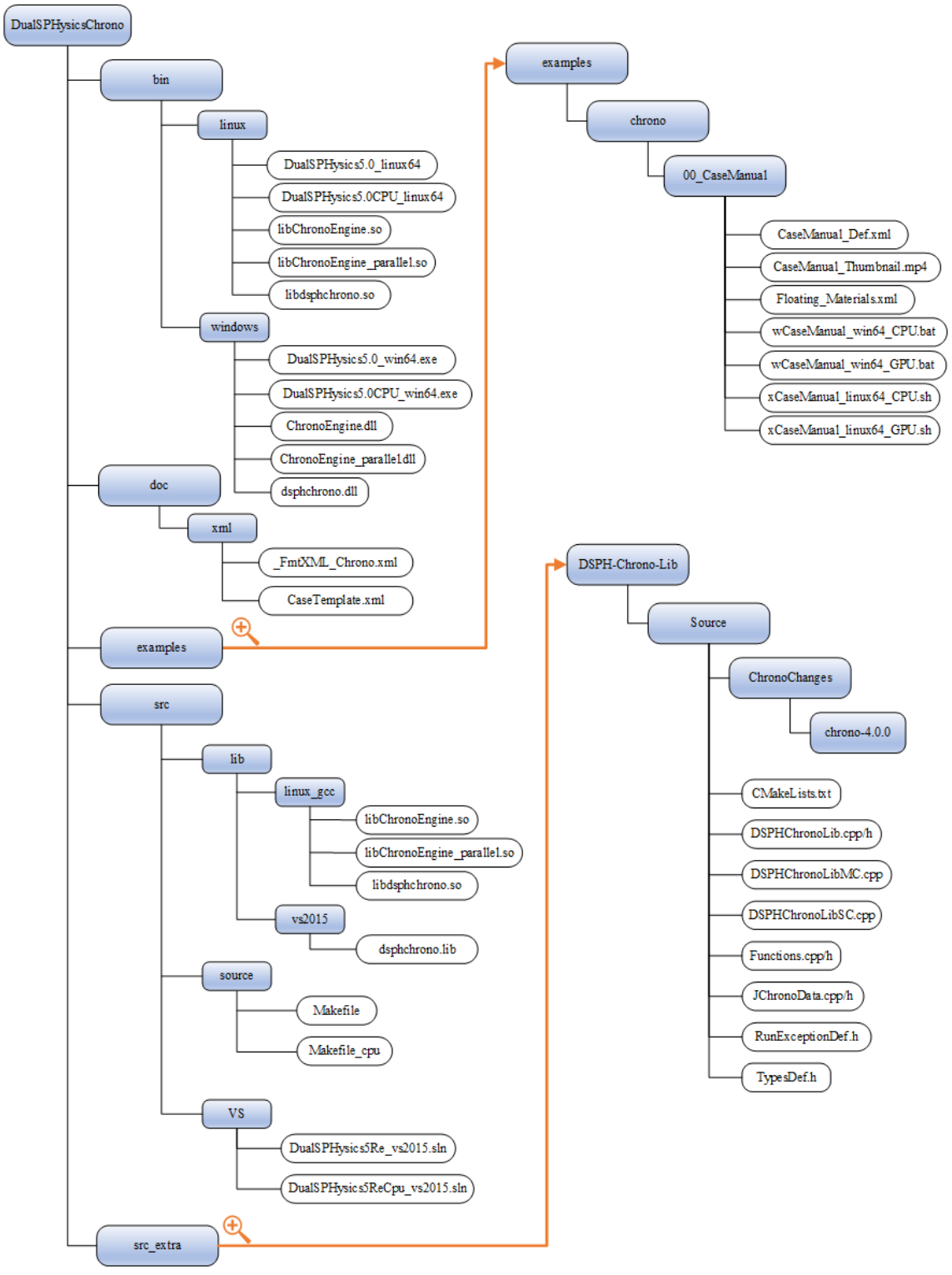

Figura 60. Estructura del directorio DualSPHysicsChrono

ANEXO 3. COMPILACIÓN DE CHRONO

El código de Chrono se puede compilar únicamente con el módulo base para ejecuciones secuenciales o añadiendo el módulo *PARALLEL* para ejecuciones paralelas.

El módulo *PARALLEL* hace uso de tres librerías adicionales.

- Blaze v3.2 (https://bitbucket.org/blaze-lib/blaze/).
- Boost v1.71.0 (https://www.boost.org).
- Thrust versión 1.9.3 o superior (https://github.com/thrust/thrust). No será necesario descargar esta librería en caso de tener instalado NVIDIA-Cuda toolkit en el equipo.

El código de Chrono está preparado para la generación y construcción del sistema usando CMake (https://cmake.org/). Las dependencias para su construcción son las siguientes:

- CMake versión 3.0.0 o superior.
- Microsoft Visual C++ 2015 (v140) en plataformas Windows.
- GNU G++ 4.9 o superior en plataformas Linux.
- Que exista el fichero "CMakeLists.txt" en **chrono-4.0.0**.

A continuación, se muestran los pasos para la compilación:

1. Abrir CMake (cmake-gui).
2. Introducir la ruta **chrono-4.0.0** en el campo "Where is the source code".
3. Introducir la ruta **chrono-4.0.0_build** en el campo "Where to build the binaries".

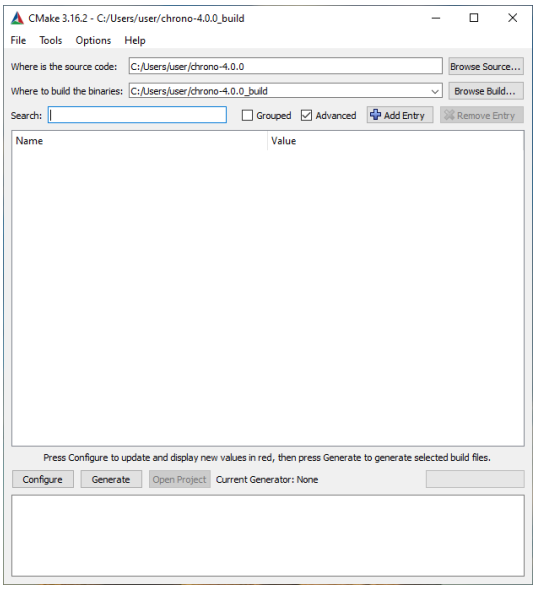

Figura 61. Introducir las rutas para la generación de Chrono

4. Pulsar en "Configure".
5. Seleccionar "Visual Studio 14 2015" y la plataforma "x64" para Windows (Figura 62) y "Unix Makefiles" para Linux (Figura 63).

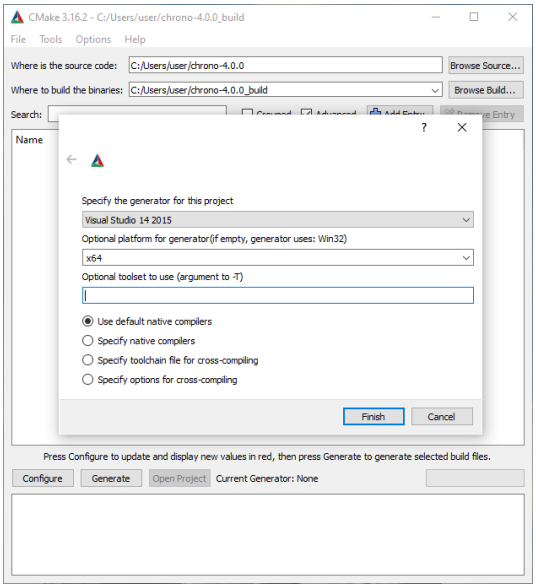

Figura 62. Selección de compiladores en Windows para Chrono

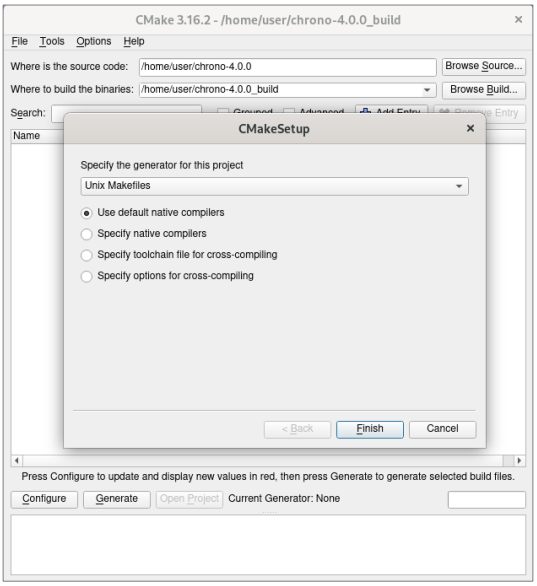

Figura 63. Selección de compiladores en Linux para Chrono

6. Introducir la ruta de **chrono-4.0.0_build/cmake** "CMAKE_INSTALL_PREFIX".

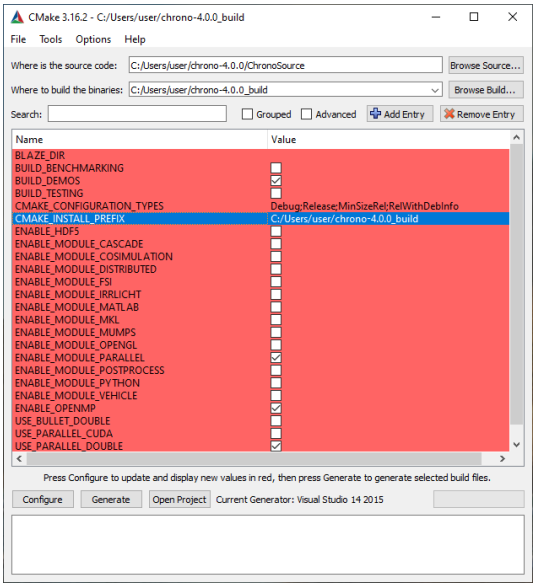

Figura 64. Seleccionar la ubicación de la configuración de CMake

7. Habilitar el uso del módulo *PARALLEL* y de la tecnología OpenMP.

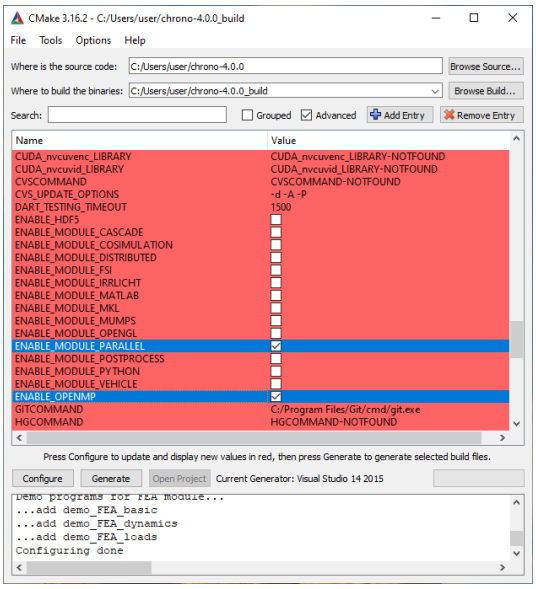

Figura 65. Habilitar el uso del módulo PARALLEL

8. Pulsar en “Configure”.
9. Introducir la ruta del directorio **blaze-3.2** en el campo “BLAZE_DIR”, la ruta del directorio **boost_1_71_0** en el campo “BOOST_ROOT”. Si no se tiene instalado NVIDIA-Cuda toolkit, introducir la ruta del directorio **thrust-1.9.3** en el campo “THRUST_INCLUDE_DIR”.

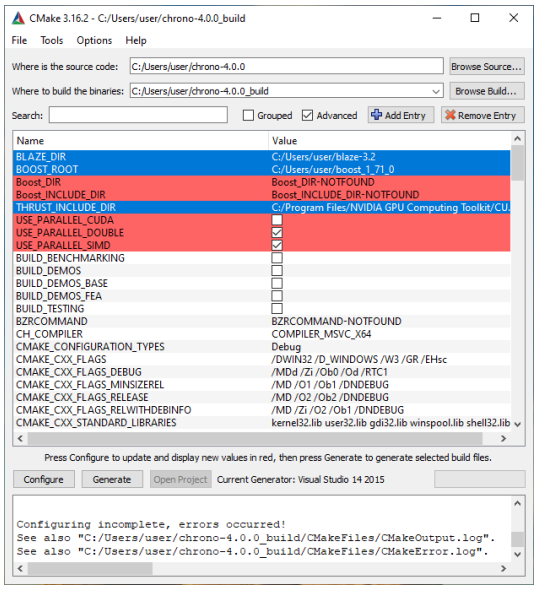

Figura 66. Introducir las rutas de la librería Blaze, Boost y Thrust

10. Pulsar en “Configure”.
11. Pulsar en “Generate” y en la ventana inferior de CMake deberá mostrarse “Generating done”.

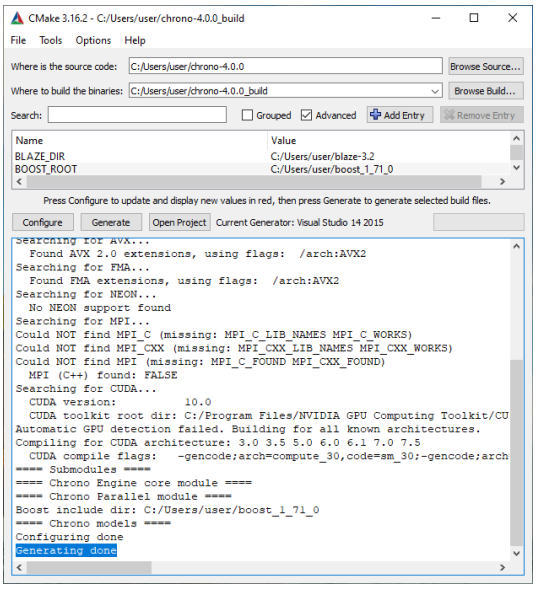

Figura 67. Generación de la solución de Chrono

WINDOWS

Si los pasos anteriores se realizaron correctamente, se generará una solución denominada **Chrono.sln** en **chrono-4.0.0_build**.

12.1. Abrir la solución con Microsoft Visual Studio 2015.

13.1. Seleccionar modo de compilación “Release”.

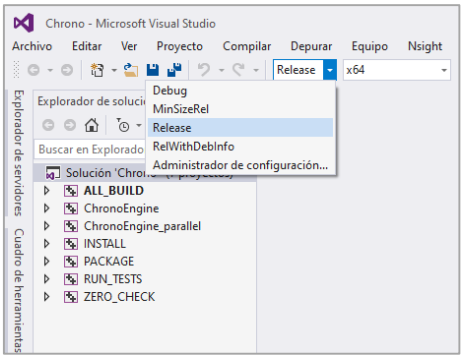

Figura 68. Selección del modo de compilación de Chrono

14.1. Pulsar en “Compilar” y a continuación, en “Compilar solución”.

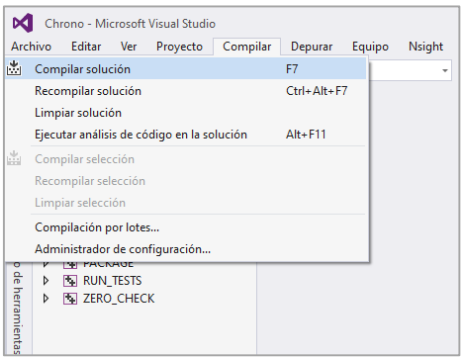

Figura 69. Compilar Chrono con Microsoft Visual Studio 2015

Al finalizar la compilación se generarán los ficheros en **ChronoEngine.dll** y **ChronoEngine_parallel.dll** en **chrono-4.0.0_build/bin/Release**. Copiar ambos ficheros en **DualSPHysicsChrono/bin/windows**.

LINUX

Si los pasos anteriores se realizaron correctamente, se generará un fichero *Makefile* en el directorio **chrono-4.0.0_build**.

11.2. Abrir un terminal de Linux en el directorio **chrono-4.0.0_build**.

12.2. Compilar el código usando la herramienta Make.

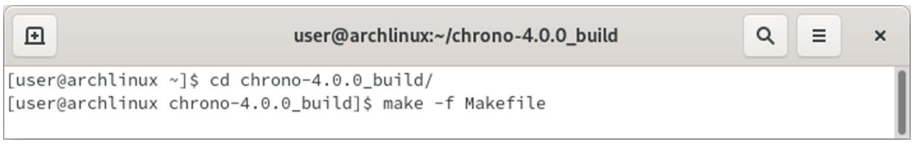

Figura 70. Compilar Chrono con Make en Linux

Al finalizar la compilación, se generarán los ficheros **libChronoEngine.so** y **libChronoEngine_parallel.so** en la ruta **chrono-4.0.0_build/lib**. Copiar ambos ficheros en los directorios **DualSPHysicChrono/bin/linux** y **DualSPHysicChrono/src/lib/linux_gcc**.

ANEXO 4. COMPILACIÓN DE DUALSPHYSICS

DualSPHysics se puede compilar para ser ejecutado en CPU y CPU&GPU. Es necesario tener instalados los drivers de **Cuda 9.2** o posterior para la versión de GPU. En caso de no tenerlos, solamente se podrá compilar y ejecutar la versión de CPU.

En DualSPHysics existe un fichero denominado *DualSphDef.h* que permite habilitar/deshabilitar el uso de librerías externas mediante definiciones de Macros. Es necesario comentar la línea **#define DISABLE_CHRONO** para compilar DualSPHysics con Chrono y para usar el módulo *PARALLEL* es necesario comentar la línea **#define DISABLE_CHRONO_OMP**, tal como se puede observar en la Figura 71.

```
DualSphDef.h
//#define DISABLE_CHRONO     ///<It allows compile without ChronoLib library (dsphchrono.dll, ChronoEngine.dll and ChronoEngine_parallel.dll)
//#define DISABLE_CHRONO_OMP ///<It allows compile without parallel module of Chrono (ignores ChronoEngine_parallel.dll).
//#define DISABLE_WAVEGEN    ///<It allows compile without Wave-Paddles, Multi-Layer Pistons and Relaxation Zones libraries.
//#define DISABLE_MOORDYN    ///<It allows compile without LibDSphMoorDyn library.  //<vs_moordyyn>
```

Figura 71. Habilitar/deshabilitar el uso de librerías externas en DualSPHysics

WINDOWS

El fichero **dsphchrono.lib** debe estar ubicado en el directorio **DualSPHysicsChrono/src/lib/vs2015** para compilar DualSPHysics con Chrono.

Es necesario tener instalado Microsoft Visual Studio 2015 y que la solución **DualSPHysics5Re_vs2015.sln** esté ubicada en el directorio **DualSPHysicsChrono/src/VS**.

1. Abrir la solución con Microsoft Visual Studio 2015.
2. Seleccionar modo de compilación.
   2.1. Modo “Release” para la versión CPU&GPU (Figura 72).
   2.2. Modo “ReleaseCPU” para la versión CPU (Figura 73).

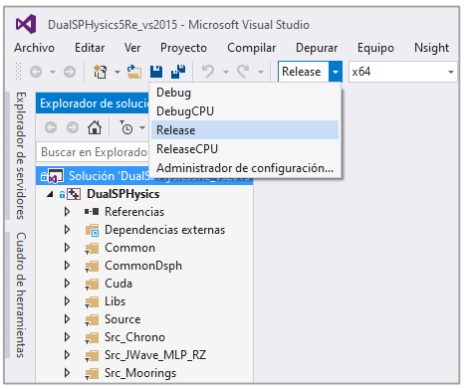

Figura 72. Selección del modo de compilación para CPU&GPU en DualSPHysics

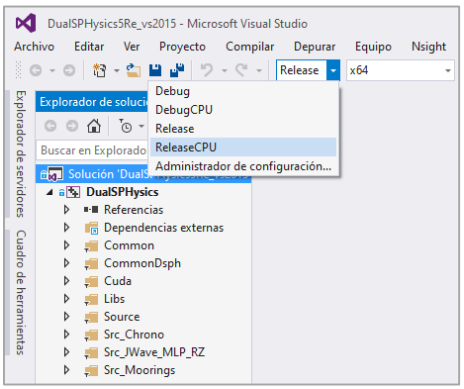

Figura 73. Selección del modo de compilación para CPU en DualSPHysics

3. Pulsar en “Compilar” y a continuación, en “Compilar solución”.

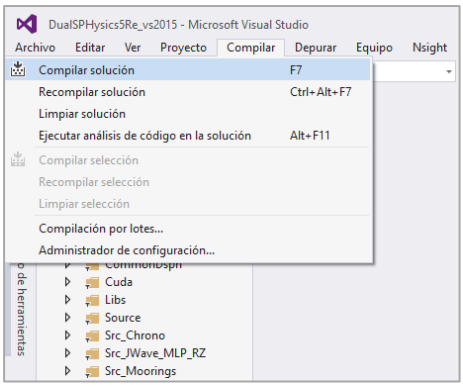

Figura 74. Compilar DualSPHysics con Microsoft Visual Studio 2015

Al finalizar la compilación se generará el fichero **DualSPHysics5.0_win64.exe** en el caso de compilar en modo "Release" y el fichero **DualSPHysics5.0CPU_win64.exe** en caso de compilar en modo "ReleaseCPU" en el directorio **DualSPHysicsChrono/bin/windows**.

LINUX

Los ficheros **libChronoEngine.so**, **libChronoEngine_parallel.so** y **libdsphchrono.so** deben estar ubicados en el directorio **DualSPHysicsChrono/src/lib/linux_gcc** para compilar DualSPHysics con Chrono.

Es necesario tener instalados los compiladores de G++ y la herramienta Make. Además, en caso de compilar la versión para GPU, es necesario tener instalado el compilador NVCC. Los ficheros *Makefile* y *Makefile_cpu* deben estar ubicados en **DualSPHysicsChrono/src/source**.

1. Abrir un terminal de Linux en el directorio **DualSPHysicsChrono/src/source**.
2. Compilar el código usando Make.
   2.1. Para CPU&GPU.

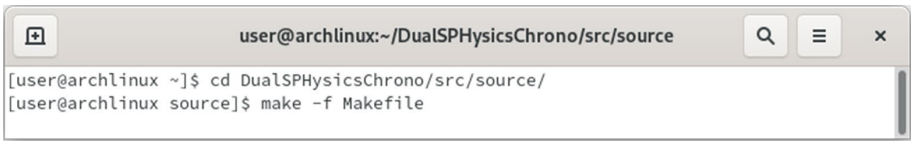

Figura 75. Compilar DualSPHysics para CPU&GPU con Make en Linux

   2.2. Para CPU.

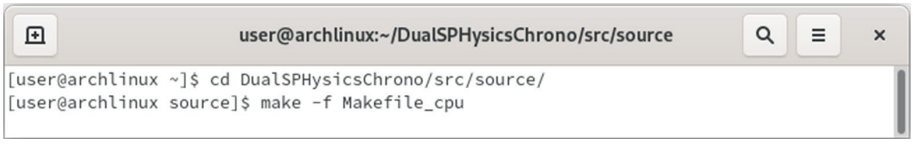

Figura 76. Compilar DualSPHysics para CPU con Make en Linux

Al finalizar la compilación se generará el fichero **DualSPHysics5.0_linux64** en el caso de compilar la versión CPU&GPU y el fichero **DualSPHysics5.0CPU_linux64** en caso de compilar en la versión CPU en el directorio **DualSPHysicsChrono/bin/linux**.