Simulation of conforming contact in real-time multibody dynamics using a volumetric force model

David Vilela Freire

Advisors:
Manuel González Castro
Alberto Luaces Fernández

Ferrol, September 2018

Dr. Manuel González Castro, Doctor por la Universidade da Coruña, y Dr. Alberto Luaces Fernández, Doctor por la Universidade da Coruña, certifican que la presente memoria, titulada *Simulation of conforming contact in real-time multibody dynamics using a volumetric force model*, ha sido desarrollada por David Vilela Freire bajo su supervisión para optar al grado de Doctor con mención Internacional por la Universidade da Coruña.

Dr. Manuel González Castro, Doctor by the University of A Coruña, and Dr. Alberto Luaces Fernández, Doctor by the University of A Coruña, certify that this doctoral dissertation, titled Simulation of conforming contact in real-time multibody dynamics using a volumetric force model, has been developed by David Vilela Freire under their supervision in order to obtain the International Doctor mention by the University of A Coruña.

Ferrol, 2018.

David Vilela Freire Doctorando *PhD. student*	Dr. Manuel González Castro Director *Advisor*	Dr. Alberto Luaces Fernández Director *Advisor*

Miguel Ángel Naya Villaverde
Tutor
Tutor

A mi familia: María, Antonio, Rubén y Sara.

Agradecimientos

Al volver la vista atrás, me resulta prácticamente incomprensible la celeridad con la que ha transcurrido este tiempo. Han pasado ya cuatro años y medio desde el comienzo de los desarrollos que han dado como fruto este documento, y es ahora cuando uno reflexiona y comprende que no son solamente las ideas, sino también las personas, las que aportan el valor que un trabajo como éste encierra. Es por ello que me gustaría, antes de comenzar, mencionar a todas esas personas que han proporcionado, de una forma u otra, algo valioso que ha quedado reflejado en estas páginas.

En primer lugar, me gustaría agradecer a mis directores, Manuel González y Alberto Luaces, la orientación que me han dado durante el progreso de esta tesis y los innumerables traspiés que he dado a lo largo de la misma. De no haber sido por la capacidad crítica de Manuel y los aparentemente inagotables conocimientos multidisciplinares de Alberto, seguramente éste habría sido un trabajo muy diferente.

En segundo lugar, querría expresar también mi gratitud hacia Javier Cuadrado y Daniel Dopico, que han apuntalado constantemente los cimientos que han sostenido esta empresa, y cuyo apoyo, especialmente durante estos últimos meses, ha hecho posible que fructifique.

Además, desarrollar un trabajo así no sería posible sin la colaboración de todas las personas que forman el Laboratorio de Ingeniería Mecánica: Miguel, Urbano, Emilio, Fran González, Amelia, Fran Mouzo, Florian, Antonio, Sarath y Borja. Ellos han creado el entorno de trabajo óptimo, tanto a nivel profesional como personal.

Mencionar también a Gabriel Zachmann y Rene Weller, del Department for Computer Science de la Universidad de Bremen, por los conocimientos que me brindaron durante mi estancia con ellos.

Por último, ni aun empleando todas las páginas restantes sería suficiente para poner en valor el apoyo incondicional de mi familia: María, Antonio, Rubén y Sara. Vosotros habéis sido la semilla y el sustrato.

Abstract

Simulation is a tool on the rise, especially in the industrial field. The usage of simulators grants the ability of studying, predicting and improving the behavior of a system, as well as designing a new one. In the case of mechanical processes simulators, the characterization of contacts and collisions between the different elements at play is one of the key factors to achieve a realistic simulation. If, furthermore, the simulator is designed to interact with machines or people, the need of real-time execution is imposed. Usually, these requirements produce a conflict of interest, since more complex algorithms demand larger execution times. Furthermore, all this is worsened by some application's need of conforming contact simulation, this is, complex contacts where the size of the contact footprint is not negligible compared to the size of the bodies in collision. This work studies two methods suitable for conforming contact simulation and their possibilities to be used in real-time simulators are discussed.

Resumo

A simulación é unha ferramenta en auxe, especialmente no ámbito industrial. O emprego de simuladores otorga a capacidade de estudar, predecir e mellora-lo comportamento dun sistema, así como de deseñar un novo. No caso dos simuladores de procesos mecánicos, a caracterización do contacto e das colisións entre os diferentes elementos en xogo é un dos factores clave para conseguir unha simulación fidedigna. Se, ademáis, ésta está deseñada para interactuar con máquinas ou persoas, imponse a necesidade de que a execución da simulación sexa en tempo real. Xeralmente, estos requerimentos producen un conflicto de intereses, xa que algoritmos máis complexos esixen tempos de execución máis amplos. Ademáis, todo isto vese perxudicado pola necesidade dalgunhas aplicacións de simular contactos conformes, isto é, contactos complexos nos que o tamaño da pegada de contacto non é desprezable en comparación ó tamaño dos corpos en colisión. Neste traballo estúdianse dous métodos adecuados para simular contactos conformes e debátense as súas posibilidades para ser aplicados en simuladores en tempo real.

Resumen

La simulación es una herramienta en auge, especialmente en el ámbito industrial. El empleo de simuladores otorga la capacidad de estudiar, predecir y mejorar el comportamiento de un sistema, así como de diseñar uno nuevo. En el caso de los simuladores de procesos mecánicos, la caracterización del contacto y las colisiones entre los diferentes elementos en juego es uno de los factores clave para conseguir una simulación fidedigna. Si, además, ésta está diseñada para interactuar con máquinas o personas, se impone la necesidad de que la ejecución de la simulación sea en tiempo real. Generalmente, estos requerimientos producen un conflicto de intereses, ya que algoritmos más complejos exigen tiempos de ejecución más amplios. Además, todo esto se ve perjudicado por la necesidad de algunas aplicaciones de simular contactos conformes, esto es, contactos complejos en los que el tamaño de la huella de contacto no es despreciable en comparación al tamaño de los cuerpos en colisión. En este trabajo se estudian dos métodos adecuados para simular contactos conformes y se debaten sus posibilidades para ser aplicados en simuladores en tiempo real.

Resumen extendido

Introducción

Desde el amanecer de los tiempos, el desarrollo de herramientas ha sido una de las características más icónicas del ser humano. La capacidad de construir, usar y mejorar cosas nos ha seguido a través de la historia hasta la Era Digital y de la Información, ayudándonos a allanar el camino para las generaciones venideras.

Durante mucho tiempo, nuestras herramientas eran simples y rudimentarias, pero la capacidad para compartir nuestras invenciones con nuestros semejantes y permitirles desarrollar nuevo conocimiento sobre el nuestro ayudó a mantener la tecnología en un ciclo de constante mejora que ha llevado a los humanos más allá de la Tierra. La forma clásica de crear una nueva herramienta o mejorar una ya existente es construir un prototipo con alguna funcionalidad nueva, probarla y corregir todos los errores de diseño detectados. Esta forma de proceder es bastante efectiva, pero también consume mucho tiempo, ya que muchas veces se necesita una reconstrucción total del prototipo para resolver un error de diseño. Afortunadamente, el uso generalizado de los ordenadores ha traído una posibilidad interesante para sortear este desafío: crear y probar máquinas inexistentes a través de la simulación en realidad virtual.

La simulación por ordenador y la realidad virtual han sido florecientes campos de investigación durante las últimas décadas. La continua caída en el precio de la potencia de cálculo ha hecho posible una explosión en la investigación relacionada con la informática, y estos campos han salido especialmente beneficiados. Desde una perspectiva empresarial, la simulación de máquinas y mecanismos conlleva una gran cantidad de ventajas competitivas y un importante ahorro en los costes: las plantas de fabricación pueden reducir ampliamente el número de prototipos fabricados, ya que sus homólogos virtuales pueden hacer frente a muchas de las pruebas necesarias previas a la producción, las instalaciones de entrenamiento pueden comenzar a preparar a sus operarios de maquinaria en entornos virtuales reduciendo en gran medida los costes y riesgos asociados e impulsando el proceso de aprendizaje, y las líneas de ensamblaje pueden detectar los posibles errores de producción más rápidamente a través de simulaciones de ensamblaje virtual. Estos

son sólo algunos ejemplos del enorme potencial que la introducción de la simulación en la industria ha involucrado.

Muchos de estos simuladores ampliamente extendidos no necesitan contar con una gran precisión física: en ocasiones es suficiente tener todos los objetos simulados en su lugar e incluso algunos de ellos no precisan de interacción con el usuario. En estos casos, el uso de modelos reducidos o simplificados implica simuladores con menores requerimientos de potencia de cálculo y que son lo suficientemente buenos para hacer el trabajo. Tomemos como ejemplo un simulador de conducción empleado para enseñar a los estudiantes que desean obtener su permiso de conducir. Para esta tarea, el simulador puede obviar la mayor parte de la complejidad mecánica de un vehículo y centrarse en el volante, la palanca de cambios y los pedales. Eso es suficiente para que el estudiante aprenda los conceptos básicos antes de subirse a un automóvil real.

Pero, ¿y si el usuario realmente necesita sentir fuerzas de retroalimentación, como es el caso de los operarios entrenándose en un simulador de grúa o los trabajadores de una línea de montaje virtual? Estos casos exigen simulaciones más realistas y precisas y las implementaciones simplificadas no se adaptan a sus necesidades. Las simulaciones de máquinas complejas deben lidiar con un alto número de entradas y la interacción entre las entidades simuladas plantean un interesante desafío de ingeniería para poder obtener una caracterización realista. Específicamente, la descripción de colisiones e impactos entre objetos es un factor clave en el desarrollo de un simulador físicamente correcto. Estas estrictas condiciones son agravadas si la simulación exige ejecución en tiempo real ya que en este caso existen restricciones de tiempo y se requieren algoritmos eficientes.

La dinámica multicuerpo es una disciplina relativamente joven que permite el cálculo del movimiento de sistemas mecánicos definidos como un conjunto de las partes interconectadas que componen dicho sistema. Una vez que cada parte y cada articulación es definida en una familia de coordenadas preseleccionada, una formulación multicuerpo integra las ecuaciones del movimiento en el tiempo, obteniendo así la dinámica del mecanismo.

Este trabajo documenta el uso de la dinámica multicuerpo combinado con un modelo de fuerza de contacto volumétrico en tiempo real.

Objetivos

El propósito de este trabajo puede ser resumido en los siguientes puntos:

- Implementar un modelo de contacto volumétrico capaz de caracterizar de forma precisa las fuerzas que aparecen en situaciones de contacto conforme.

- Investigar diferentes enfoques para calcular los volúmenes de intersección en colisiones simuladas de forma genérica.

- Evaluar un conjunto de ejemplos y estudiar la viabilidad de la implementación del método para procesar simulaciones en tiempo real.

Estructura de la tesis

Esta tesis está dividida en los siguientes seis capítulos:

El capítulo 1 es una breve introducción a la motivación que desembocó en la creación de este documento, y en él se presenta también el estado del arte de la dinámica multicuerpo y los últimos desarrollos en modelos de contacto y detección de colisiones.

El capítulo 2 describe la disciplina de la dinámica multicuerpo empleada para las simulaciones y la forma en que se formulan e integran las ecuaciones del movimiento para recrear el movimiento de un mecanismo. Se concluye que la formulación de Lagrange Aumentado con proyecciones en índice 3 satisface los requerimientos de estabilidad y eficiencia que precisa un simulador interactivo. Esta formulación permite resolver el sistema diferencial-algebraico de las ecuaciones del movimiento para cada instante de tiempo.

En el capítulo 3 se discute un modelo de fuerzas de contacto basado en el modelo volumétrico de Gonthier, que sirve para caracterizar las fuerzas normales y tangenciales que aparecen durante la colisión entre dos cuerpos. Éste modelo es capaz de calcular dichas fuerzas a partir del volumen de intersección entre objetos con forma arbitraria, de manera que constituye un método general para modelar los contactos de forma eficiente y precisa.

El capítulo 4 trata sobre los métodos de detección de colisiones y sobre cómo calcular los parámetros necesarios para caracterizar los impactos. Se presentan dos algoritmos de detección de colisiones, uno basado en superficies de mallas de triángulos y otro en aproximaciones volumétricas de esferas, cada uno con sus particularidades. Estos métodos posibilitan identificar situaciones de contacto y calcular la geometría de la intersección entre los objetos en colisión que posteriormente se emplea para alimentar al modelo de fuerzas de contacto.

El capítulo 5 muestra la metodología presentada aplicada a tres casos diferentes y sus resultados.

En el capítulo 6 se presentan las conclusiones obtenidas de este trabajo y se proponen futuras líneas de investigación.

Metodología

Para verificar la validez de los modelos de contacto implementados, se diseñaron tres tests diferentes orientados cada uno de ellos a un tipo específico de movimiento relativo entre cuerpos y su correspondiente fricción: deslizamiento, giro y rodadura.

Test 1: bloque deslizando en un plano inclinado

Este test consistió en un bloque de 1 m x 0.5 m x 0.5 m y 1 kg de masa deslizando cuesta abajo de manera que su mayor dimensión coincide con la dirección de deslizamiento. El objetivo del test era comparar el ángulo a partir del cual la fuerza de rozamiento no puede mantener el bloque estático y éste comienza a deslizar. El valor teórico para un coeficiente de fricción $\mu = 0.5$ es 26.5650°. Además, se ejecutó una simulación de 4 segundos de duración con un plano inclinado 30° para asegurar el deslizamiento y comprobar la trayectoria del centroide del bloque, su velocidad y la desviación de sus ejes con respecto a su orientación inicial.

Para el modelo de malla, la desviación comenzó a los 26°, lo cual implica una precisión del 97.87 %. El error en la trayectoria del centroide se mantuvo por debajo de 5×10^{-6} durante la simulación de 4 segundos mostrando un crecimiento cuadrático. El perfil de velocidad resultó ser prácticamente idéntico al teórico y el mayor desvío angular fue de -6×10^{-5}. Estos resultados muestran una precisión muy cercana a la ideal.

El modelo de esferas o IST de este test estuvo compuesto por 12 215 para el bloque y 15 423 para el suelo. A pesar de que aparentemente el contar con cientos de fuerzas de contacto en lugar de una sola debería de hacer que la caracterización de las colisiones se asemejase más a una distribución de fuerzas, el modelo de stiction encontró dificultades para manejar todas esas colisiones simultáneas, y no pudo detener al bloque completamente, quedando en todas las pruebas una velocidad residual dependiente del ángulo del plano base. Los errores de trayectoria, velocidad y desviación angular fueron bastante significativos y dependientes de la posición inicial.

En cuanto al rendimiento, el modelo de malla se ejecutó unas 19 veces más rápido que el modelo IST, siendo sólo el primero capaz de alcanzar la ejecución en tiempo real.

Test 2: disco rotando en un plano

Se diseñó un test en el que un disco de radio $R = 0.25\,\mathrm{m}$, altura $H = 0.05\,\mathrm{m}$ y masa $m = 1\,\mathrm{kg}$ rota sobre un plano a una velocidad angular $\omega_0 = 5\pi\,\mathrm{rad\,s^{-1}}$ (2.5 Hz). El coeficiente de fricción fue $\mu = 0.6$. El objetivo de este test fue validar el modelo de resistencia al giro, y se emplearon unos parámetros equivalentes al

test anterior, además de medir el tiempo que el disco tardó en detenerse debido a la fricción. El valor teórico para este tiempo de frenado es $t = 0.5\,\mathrm{s}$.

El modelo de malla tardó en detener el disco 1 s. La fuerza de frenado fue continua y suave aunque imprecisa. El error de trayectoria máximo fue de 1×10^{-4} y los errores de velocidad y desviación angular fueron despreciables.

Para el modelo de esferas se emplearon un IST de 11 212 esferas para el disco y otro de 21 272 para el suelo. El perfil de frenado fue muy similar al teórico, siendo el tiempo de parada solamente 0.07 s menor al esperado. El error de trayectoria fue de 9 mm, el de velocidad 0.05 m/s y el de desviación angular 0.0022 rads. En general, la simulación mostró bastantes vibraciones debido a las colisiones entre esferas.

Ninguno de los dos modelos consiguió alcanzar el tiempo real, si bien el modelo de malla se aproximó bastante, ejecutándose 19 veces más rápido que el IST.

Test 3: cilindro rodando en un plano inclinado

Para comprobar la calidad del modelo de resistencia a la rodadura se empleó un test consistente en un cilindro de radio $R = 0.25\,\mathrm{m}$, altura $H = 1\,\mathrm{m}$ y masa $m = 1\,\mathrm{kg}$ rodando en un plano inclinado 15°. Se registraron, de forma análoga a los anteriores tests, los perfiles de trayectoria, velocidad y orientación para compararlos a los teóricos.

El modelo de malla mostró unos perfiles muy similares a los esperados, con errores prácticamente despreciables durante el primer segundo de simulación. Solamente el error de velocidad aumentó a partir de ese punto hasta alcanzar un desvío del 8 % con respecto al teórico. Mayores inclinaciones de plano implicaron un aumento en la frecuencia del ruido debido a la modelización del cilindro mediante una malla de triángulos.

En cuanto al modelo de esferas, se empleó un IST de 8849 esferas para el cilindro, mientras que para el suelo hubo que generar una disposición de 200×30 esferas alineadas, ya que el IST inicial produjo trayectorias incorrectas durante las simulaciones, llegando a desviar el cilindro perpendicularmente a su trayectoria. Con la nueva modelización, el error máximo de trayectoria fue de 1 cm, el desvío angular de 1° y el perfil de velocidades se aproximó bastante al teórico.

El modelo de malla se ejecutó 6.5 veces más rápido, siendo el único en alcanzar el tiempo real.

Conclusiones y trabajo futuro

Esta tesis tiene por objetivos evaluar la factibilidad del empleo de modelos de fuerza volumétricos en situaciones de contacto conforme en entornos de tiempo

real e investigar la precisión de dichos métodos.

La unión entre la formulación multicuerpo, el modelo volumétrico de contacto y el método de detección de colisiones hizo posible el desarrollo de un algoritmo capaz de simular contactos conformes en tiempo real, y por lo tanto es apto para ser empleado en entornos interactivos como simuladores de ensamblaje virtual. Las ecuaciones del movimiento fueron expresadas en una formulación Lagrangiana Aumentada de índice 3 e integrada con un paso de tiempo de 1 milisegundo a través de la regla trapezoidal. Se implementó un modelo volumétrico de contacto de Gonthier para calcular las fuerzas normales y se acopló a un modelo de fuerzas tangenciales para modelar los fenómenos de fricción. Dos algoritmos de detección de colisiones diferentes, uno basado en superficies de mallas de triángulos y otro en aproximaciones volumétricas de esferas, fueron empleados para calcular las propiedades de intersección durante las colisiones y alimentar al modelo de Gonthier. Para el modelo de malla, se escribió una librería de detección de colisiones (LIMCODE). Esta librería calcula intersecciones malla-malla y fue desarrollada con la coherencia de resultados en mente.

Se diseñaron e implementaron tres tests diferentes para investigar las fortalezas y debilidades de los dos enfoques en el cálculo de los volúmenes de intersección. Estos tests tenían por objetivo validar la precisión del modelo de fuerzas en todo tipo de situaciones de contacto comunes presentes en simulaciones de ensamblaje virtual, y cada uno de ellos fue orientado a uno de los posibles movimientos relativos entre dos cuerpos: deslizamiento, rodadura y giro.

Sobre nuestra librería de dinámica multicuerpo (MBSLIM) y nuestras librerías de detección de colisiones (MBSMODEL, LIMCODE), se implementó una librería de exportación de datos (MBSDEBUG) en C++ para depurar y visualizar colisiones, recortes de triángulos y procesos de reconstrucción de mallas, e identificar los posibles escollos para estos algoritmos. Esta librería exporta información gráfica en el formato EnSight6 que puede ser abierto con el software Paraview.

Tras evaluar los resultados de estos tests, se sacaron las siguientes conclusiones:

- El modelo de detección de colisiones basado en mallas de triángulos es capaz de producir resultados precisos en tiempo real en la mayor parte de los casos. El hecho de contar con una representación casi exacta (exacta para superficies planas) del volumen de intersección es claramente un factor clave para el cálculo de los parámetros necesarios para el modelo de fuerzas.

  Por otra banda, este modelo calcula simplemente una fuerza total. Para calcular la dirección de dicha fuerza, se debe asumir un contorno de contacto plano para poder aproximarlo a un plano de contacto y obtener su normal, lo que hace que el modelo pierda generalidad. Este problema y una posible solución se explicarán y discutirán más en profundidad en la próxima sección.

- La implementación del modelo de esferas o IST, a pesar de no ser capaz de ser ejecutada en tiempo real o generar simulaciones realistas, es mas sencilla y general, ya que reduce una colisión volumétrica compleja a un número de contactos esfera-esfera y consecuentemente a un número de fuerzas, asemejándose más, por lo tanto, a una distribución de fuerzas. A pesar de ello, la aproximación hecha al emplear este modelo parece ser demasiado tosca para representar fielmente colisiones de ensamblaje virtual, al menos de una manera realista.

- En los test ejecutados, con el número seleccionado de esferas para el modelo IST, el modelo de malla se ejecutó hasta 20 veces más rápido. Probablemente la evaluación del IST sería más rápida con menos esferas, pero esto desafortunadamente implicaría perder precisión y realismo en la simulación, que con entre 10000 y 15000 es ya insuficiente para cumplir los requisitos, como se ha visto anteriormente. Además, la necesidad de rellenar los objetos con esferas implica que, en objetos con un ratio superficie/volumen bajo, como los empleados en los tests, muchas de las esferas no se ven involucradas en las colisiones ya que son esferas internas a mayor profundidad que la máxima penetración. Esto causa una gran ineficiencia desde el punto de vista de la memoria, ya que la información de todas las esferas es cargada pero no empleada en su mayoría. El problema empeora debido al hecho de que el modelo de stiction necesita llevar un registro del estado de las colisiones en el paso de tiempo anterior, y por lo tanto incrementa la cantidad de información que debe ser guardada para cada par de esferas. Esto exige que en cada paso de tiempo haya que realizar una búsqueda a través de una lista de pares de esferas activas que resulta bastante costosa computacionalmente.

Existen diversas líneas de investigación que podrían mejorar sensiblemente el estado actual de este trabajo.

Generalización de las colisiones

A pesar de que los test descritos en esta tesis cubren algunas situaciones que pueden aparecer en condiciones de contacto conforme simples, deberían probarse nuevos tests para validar el rendimiento de los métodos en situaciones más complejas, como un cubo (o cualquier otro objeto) insertado en un agujero donde encaja perfectamente. Este tipo de test implica contacto conforme multi-plano y sería interesante estudiar cómo se comporta el modelo volumétrico en estas situaciones. Además, se deberían probar colisiones generales en las que aparezcan diversos tipos de fricción simultáneamente para asegurar que los modelos de fricción no interfieren los unos con los otros.

También se deberían realizar tests con un mayor número de objetos para estudiar cómo afecta el tamaño del problema a la velocidad de ejecución. Se supone que una simulación de realidad virtual contará con múltiples objetos interactuando, así que debe asegurarse un buen rendimiento con un alto número de mallas.

Finalmente, contar con un simulador de ensamblaje virtual sería el entorno de pruebas ideal para probar todo tipo e colisiones y validar la aplicabilidad de este tipo de métodos para producción.

Optimizaciones

Aparte de las optimizaciones puramente de programación que siempre pueden ser realizadas para aumentar el rendimiento, algunos cambios podrían también aportar mejoras a los tiempos de simulación.

- Incrementar el paso de tiempo: actualmente se está empleando un paso de tiempo de 1 milisegundo, pero podría ser aumentado a 3, 5 o incluso 10 milisegundos siempre y cuando la precisión no se vea afectada. Visualmente, los usuarios no percibirán frecuencias de refresco mayores a 60 fotogramas por segundo, de manera que aumentar el paso de tiempo podría permitir la inclusión de un número mayor de objetos interactivos en la simulación manteniendo una visualización lo suficientemente buena. A pesar de ello, si se conecta algún dispositivo háptico a la simulación, este cambio ha de ser tratado con cuidado ya que la frecuencia de refresco óptima para éstas máquinas es de 1KHz.

- Llamar a la rutina de detección de colisiones una vez por paso de tiempo: en algunos pasos de tiempo, especialmente en aquellos en los cuales los objetos están en colisión o durante contactos complejos, el integrador debe iterar varias veces hasta converger a una solución. Esto conlleva desplazamientos muy pequeños de los objetos que generalmente no implican cambios en las mallas de colisión. A pesar de esto, la detección de colisiones es llamada incondicionalmente en cada iteración. Al ser la tarea con mayor uso de CPU, eliminar estas llamadas extra podría suponer un notable ahorro de tiempo de ejecución.

Paralelización

Los métodos presentados en este documento se ejecutan secuencialmente, uno tras el otro. Las arquitecturas de los microprocesadores modernos, incluso las de los de uso doméstico, cuentan con la capacidad de ejecutar varias tareas simultáneamente. Dividir un algoritmo en sub-tareas que puedan ser ejecutadas en paralelo

no es una labor trivial, ya que la sincronización entre procesos independientes tiene un coste no despreciable. Para cada tipo de problema se debe buscar un esquema de paralelización satisfactorio que minimice la transferencia de información entre procesos. En [28] se manifiestan algunas advertencias acerca del alto coste que puede suponer la paralelización de problemas multicuerpo pequeños o medianos: normalmente el cuello de botella es el cálculo del Jacobiano de las restricciones y la resolución del sistema de ecuaciones lineal final.

Una línea de investigación interesante consiste en dividir el sistema multicuerpo en varios sub-mecanismos con sus propios procesos que interactúan mediante el intercambio de fuerzas de reacción. Los costes de sincronización se pueden evitar empleando comunicaciones inter-proceso.

Detección de colisiones con tiempo crítico para el modelo IST

Un algoritmo de detección de colisiones crítico en el tiempo implicaría ajustar el cálculo de las propiedades de intersección volumétrica a una cantidad de tiempo predeterminada. En [58] se realizaron los primeros esfuerzos para implementar esta característica en la librería CollDet. Este enfoque implicaría que, durante el recorrido del IST, se devolvería un volumen de intersección aproximado en caso de que se alcance el tiempo máximo. Este comportamiento asegura una ejecución a tiempo real con el coste de perder precisión en las colisiones más complejas. Emplear este algoritmo modificado podría ser beneficioso para mantener una frecuencia de refresco interactiva, al menos para simulaciones human-in-the loop o hardware-in-the-loop.

Stiction con múltiples contactos

El modelo IST fue incapaz de detener completamente el bloque deslizante del primer test. Parece ser que el modelo de stiction empleado podría tener problemas al manejar cientos de contactos simultáneamente. Es necesaria una investigación más profunda para clarificar este comportamiento.

Descomposición de formas arbitrarias

Los modelos de fuerza necesitan un punto de aplicación y una dirección para poder ser introducidos en la simulación. Esto plantea un revés para los modelos volumétricos como el modelo de malla presentado en este trabajo: un volumen de intersección no tiene tal dirección, de manera que para burlar este problema y calcular un vector normal, se emplea el contorno de contacto para calcular un plano mediante el método de los mínimos cuadrados. La normal de este plano sirve como

dirección de la fuerza normal, pero esta forma de proceder relega el modelo a los anteriormente mencionados contactos planos, ya que las intersecciones multi-plano no son aproximables a un sólo plano y por lo tanto no satisfacen los requisitos del modelo de Gonthier.

El modelo IST no sufre este problema exactamente, ya que cada par de esferas es tratado de forma independiente y su normal se calcula fácilmente como el vector director entre los centros de ambos centros. A cambio, muchas de estas normales son imprecisas y generalmente precisan de algún tipo de filtrado para evitar comportamientos extraños. Ejemplos de esto son las colisiones entre una esfera del cuerpo A que ha penetrado complemente la capa más externa de esferas del cuerpo B y genera una fuerza de atracción cuando los dos cuerpos comienzan a separarse, o fuerzas con un gran componente tangencial debido a colisiones laterales entre esferas.

Para que el modelo de malla sea capaz de enfrentarse a colisiones no planas (como un cubo insertado en un agujero cúbico de sus mismas dimensiones), se podría aplicar una descomposición arbitraria. Los objetos serían preprocesados y divididos en un conjunto de formas convexas, y cada parte sería usada para calcular una fuerza empleando el modelo de malla presentado. De esta forma, cada parte en contacto generaría una colisión convexa-convexa individual adecuada para alimentar el modelo.

Contents

List of Figures

List of Tables

Acronyms

AABB Axis-Aligned Bounding Box. 33–35, 37, 46

AVX Advanced Vector eXtensions. 30

B-Rep Boundary Representation. 26, 27

BSP Binary Space Partitioning tree. 5, 37, 38, 42, 81

CAD Computer Aided Design. 26

CFD Computer Fluid Dynamics. 26

COM Center of Mass. 34, 42, 44

CSG Constructive Solid Geometry. 26, 28

FEM Finite Element Modeling. 3, 26, 28, 30

FPS Frames Per Second. 84

FSM Finite State Machine. 39, 40

GPU Graphics Processing Unit. 44

IST Inner Sphere Tree. 25, 45, 59, 61, 64, 68, 70, 73, 77, 79–81, 83, 85, 86

LIMCODE LIM Collision Detection. 31, 34, 40

NURBS Non-Uniform Rational B-Spline. 26–28

OBB Oriented Bounding Box. 34

SIMD Single Instruction Multiple Data. 44

SSE Streaming SIMD Extensions. 30

Chapter 1

Introduction

Since the dawn of time, tool development has been one of the most iconic features of the human nature. The ability of building, using and improving things has followed us through our history to the current Digital and Information Age, helping us to pave the way for the coming generations.

For a very long time, our tools were simple and rudimentary, but our skills in sharing our inventions with our peers and letting them build more knowledge on top of ours helped to keep technology in an ever-improving cycle that has driven humans beyond the Earth. The classic way of creating a new tool or improving an existent one is to build a prototype with some new functionality, test it and correct every design error detected. This way of proceeding is quite effective, but also time consuming, since many times a total rebuild of the prototype is needed in order to solve a design error. Fortunately, the mainstream use of computers has brought an interesting possibility to circumvent this challenge: to create and test non-existent machines through simulation in virtual reality.

Computer simulation and virtual reality have been flourishing fields of research for the last decades. The continuous decay in computing power price made possible an explosion in computer-related research, and these fields have been especially benefited. From a business perspective, the simulation of machines and mechanisms entails a lot of competitive advantages and important cost savings: the number of prototypes can be widely reduced in manufacturing plants as their virtual counterparts can cope with a lot of the testing needed before production, training facilities can start training their machinery operators in virtual environments greatly reducing the associated cost and risk and boosting the learning process, and assembly lines can detect potential production errors faster through virtual assembly simulations. These are just some examples of the huge potential that the introduction of simulation in the industry has involved.

Many of this widely spread simulators don't have the need for a great physical accuracy: sometimes it is enough to have all simulated objects in place and even

some of them don't need interaction capabilities with the user. In this cases, the use of reduced or simplified models implies less computing-power-hungry simulators which are good enough to get the job done. As an example, let's think of a driving simulator used to teach students who want to obtain their driving licenses. For this task, the simulator can obviate most of a vehicle's mechanics complexity and just focus on the wheel, gearshift and pedals. That would be sufficient for the student to learn the basics before getting into a real car.

But, what if the user really needs to feel force feedback, like those operators training in a crane simulator or workers in a virtual assembly line? This use-cases demand more realistic and accurate simulations and naive implementations do not suit their needs. Full complex machine simulations must deal with a high number of inputs and interaction between simulated entities pose an interesting engineering challenge in order to be realistically characterized. Specifically, the description of object collisions and impacts is a key factor in the development of a physically correct simulator. This strict conditions are aggravated if the simulation needs real time execution as in this case time restrictions are imposed and efficient algorithms are required.

Multibody dynamics is a relatively young discipline that allows for the computation of motion in mechanical systems defined as an interconnected group of every of the parts that compose the full system. Once every part and every joint is defined in a previously selected family of coordinates, multibody formulations integrate the equations of motion over time, thus obtaining the mechanism dynamics.

This work documents the use of multibody dynamics combined with a volumetric contact force model in order to simulate conforming contact collisions in real-time.

1.1. Motivation

Simulators and virtual reality offer benefits and opportunities for virtually every company or business involved somehow with manufacturing, prototyping, assembly, design or machine and vehicle training. Some of the most prominent strengths that this technology unleashes are:

- Fast design evaluation and prototype stage reduction
- Operator physical risk minimization
- Cost reduction through the whole product life cycle
- Assembly line early error detection

- Hardware costs reduction and risk minimization for expensive machinery
- Operator training course automated evaluation

With the increasing demand of computer-powered solutions in the industry and the emergence of a new dawn of virtual reality in the last five years that is quickly making its way into the mainstream usage, it is no secret that having the right simulation tools poses a big competitive advantage.

As previously stated, contact and impact characterization is a key issue in the process of obtaining an accurate simulation and it usually constitutes the computing bottleneck, thus the development of fast and robust algorithms is one of the most interesting areas of research.

Traditionally, two different approaches have been employed to crack this problem: Finite Element Modeling (FEM) methods, which are extremely accurate but incapable of being run in real time due to their complexity, and methods similar to those integrated in video game physics engines, that are fast but usually yield poor results from a physical point of view, as they only look for visually feasible simulations and not accurate ones.

A hybrid solution that can calculate physically accurate forces yet fast enough in order to be executed in real time simulations would open a new door full of possibilities.

1.2. State of the art

1.2.1. Multibody dynamics

Rigid-body system dynamics has been an active research topic for a very long time now, but it wasn't until the computer science early development that multibody dynamics was consolidated and took its modern shape [51]. [52] and [18] made great historical reviews of multibody dynamics evolution.

It is widely accepted that the first practical solution methodology for large multibody systems was the one presented in [46]. This research lead to the development of ADAMS (Automatic Dynamic Analysis of Mechanical systems), and was followed by a rapid growth in the use of multibody dynamics in several fields like aerospace, automotive, robotics or biomechanics [44] and its corresponding new software solutions developments [25].

The increase in the available computational power, and the development of new, more efficient formulations allowed the use of multibody dynamics in real time simulations [35] and made human-in-the-loop and hardware-in-the-loop simulations possible. Recent developments have been done towards the implementation

of real-time multibody models in low-cost hardware platforms [48] and machinery simulators [38].

1.2.2. Contact models

Carefully modeling the contact forces that appear during collisions in simulations is a key factor in order to obtain adequate results. Moreover, if the application requires to be integrated in a human-in-the-loop simulation, restrictions on the time-step, number of iterations, simulation stability and robustness are tighter and impose an aggregated difficulty.

Typically, two different approaches have been proposed to resolve the rigid body collision problem that led to two distinct method families: the discontinuous and the continuous [36, 23, 25]. The rigid body assumption implies that bodies are supposed to be hard and only small local deformations are required to generate very large contact pressures [55]. The first family, the discontinuous approach, assumes an instantaneous impact and therefore an instantaneous change in the balance momenta [14, 15]. The continuous approach, on the other hand, is based on regularized-force models that relate forces and body deformation [36, 33] or constraint techniques that avoid the penetration between bodies [40, 50, 6, 24]. The continuous methods based on regularized forces include a number of viscoelastic and viscoplastic models [25, 34, 8, 27]. Many of these continuous methods are based on the Hertzian contact theory thus assume contact areas much smaller than the characteristic dimensions of the contacting bodies. This poses a problem when trying to simulate conforming contacts.

There exist a large number of formulations of the equations of motion that describe the movement of mechanical systems [35]. Among them, the augmented Lagrangian formulations [2, 3] used in this work have the peculiarity of transforming the constraints into forces proportional to the constraints violation, which makes them compatible with the continuous force methods.

1.2.3. Collision detection

Collision detection encloses different techniques and procedures to solve an apparently simple problem: to determine if two objects are in contact. The answer to this question and its corresponding mathematical implementation, despite its seeming simplicity, is far from trivial. It is understood that two objects are colliding if they share at least one point in the space at the same time.

The first attempts to resolve this issue appeared with the first robots. The automation at the assembly lines required that robot maneuvers were simulated in order to verify the interference between machines [4]. Also, early developments

in computer graphics and the study of geometric algorithms gave birth to computational geometry. This research spawned a wide range of knowledge leading to the first methods to compute geometric properties such as polyhedra convex hulls, distance calculations or intersection detection. It was on the early 1980s, with the appearance of interactive 3D applications and the spread of home computers and video games, that collision detection as we know it today started to take shape. The first applications and games, constrained by the hardware specs, simplified collision detection using simple shapes, like spheres and boxes, to check if two objects were in contact. Further research was done in the 1990s as increasingly complex computer animations joined the area of influence of collision detection, for example to meet the requirements of the film industry.

As computer processing power grew, so did the complexity of animations and simulations, and to keep the collision detection power requirements under control, many optimized techniques were developed: spatial data structures like the Binary Space Partitioning tree (BSP), and bounding volume hierarchies based on Axis Aligned Bounding Boxes (AABB) or Oriented Bounding Boxes (OBB) were used to swiftly reject large number of geometric primitives from the intersection being tested and boost the computation. Many of these now well established algorithms are described in [4] and [22].

1.3. Objectives

The aim of this work can be summarized in the following points:

- Implement a volumetric contact model capable of accurately characterizing the forces that arise in conforming contact situations.
- Research different approaches to calculate the intersection volumes on collision simulations in a general way.
- Evaluate a set of collision test cases and assess the viability of the method implementation in order to process simulations in real time.

1.4. Summary

This thesis is divided into the following six chapters:

Chapter 1 is a brief introduction to the motivation which led to the creation of this document, presenting also a brief state of the art of multibody dynamics and the contact and collision detection last developments.

Chapter 2 describes the multibody dynamics discipline used for the simulations and the way the equations of motion are integrated to recreate a mechanism movement.

Chapter 3 discusses the contact models whose aim is to define the forces that stress the system.

Chapter 4 deals with collision detection methods and how to calculate the needed parameters in order to characterize impacts.

Chapter 5 shows the presented methodology applied to three different test cases and its results.

Chapter 6 presents the conclusions drawn from this work and proposes future lines of research.

Chapter 2

Multibody dynamics formulation

2.1. Multibody dynamics

Before the computers went mainstream, mechanical problems had traditionally been solved by hand. For this reason, most of them were simple cases that could be worked using analytical expressions. In the case of mechanism dynamics or motion analysis for more complex ones, the solving was focused on just specific configurations, whether they were worst-case scenarios or interesting situations. The classical approach consists in finding the system energy equations and solve the system imposing a force equilibrium. This is usually rendered difficult to achieve since the differential equations obtained can be really complex to solve, especially in closed-loop mechanisms.

Once the computer usage spread out across the globe, a lot of research fields were benefited from it. In particular, mechanical simulation was rendered feasible not only for specific case or configurations, but for a much broader set of problems. For this reason, many computer-oriented disciplines were born in order to take on the new possibilities: multibody dynamics was one of them.

Multibody dynamics enables the definition of a system by means of a coordinate set and the integration of the equations of motion over time. It requires to describe the problem by defining each of its parts and the joints that act as links between them. As a computerized field, simple and generalized mathematical methods are promoted over the ones that were used when solving by hand the system of equations yielded. While the variety of coordinates, formulations and integrators is very diverse, it is recommended to make a careful selection to improve the method performance.

2.1.1. Coordinates

Different families of coordinates can be used within multibody dynamics. This choice will affect the easiness in the modeling stage, the number of equations of motion and also the computation cost needed in order to integrate them. Some of these families simplify the modeling using less redundancy at the cost of creating a less flexible system definition, while others demand a bigger effort in the coordinates selection phase in exchange of a more redundant and flexible definition. Examples of this families are the following:

- Reference point coordinates
- Relative coordinates
- Natural coordinates
- Mixed coordinates

For this work, the natural family of coordinates was selected. Like the reference point coordinates, natural coordinates locate every element independently from the others, but the selected reference points match the joints location, so that every joint can be used for the definition of two consecutive elements.

2.1.2. Formulation

The multibody formulation is the final form of the equations of motion used to solve the system dynamics. This form will be affected by the coordinates selection, the mechanical principles used to derive the equations of motion and the way to enforce the constraint equations. All these choices also determine the simulation performance and are greatly influenced by the problem topology: while some formulations may yield an improved performance, others may facilitate the computation of some variables like reaction forces. In the end, every problem should be carefully analyzed and a formulation must be selected as a trade-off between performance, accuracy, stability and complexity. Some of the usual formulations include direct solving, stabilized Lagrange, Baumgarte stabilization, projection matrix R and penalty formulations, each one with its peculiarities, advantages and disadvantages.

The formulation used during this work was the *Index-3 Augmented Langrangian formulation with projection of velocities and accelerations*. This formulation combines the penalty formulation with the Lagrange multipliers method, allowing for the use of smaller penalty factors, yielding a better numerical conditioning and leading to the exact solution if the numerical implementation is appropriate. An

extra iteration loop must me performed in order to update the Lagrange multipliers, causing a very small computational overhead due to most of the terms being constant in the iteration. This formulation has been proven robust and efficient, and has been tested extensively in [1], [10] and [16].

2.2. Equations of motion

The configuration of a multibody system can be defined using n generalized coordinates, related by means of m constraint equations. When no redundant constraints are used, the system has $n-m$ degrees of freedom. Constraints described here are holonomic, kinematic constraints, expressed as $\mathbf{\Phi}(\mathbf{q},t)=0$, being $\mathbf{q}$ the vector where the n generalized coordinates are stored.

An application of the *virtual work* principle is shown in (2.1). Any *virtual displacement* has to be compatible with the constraints of the system.

$$\delta\mathbf{q}^{*\mathrm{T}}(\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}}-\mathbf{Q}(\mathbf{q},\dot{\mathbf{q}}))=\mathbf{0} \tag{2.1}$$

$$\mathbf{\Phi}(\mathbf{q},t)=\mathbf{0} \tag{2.2}$$

$\mathbf{M}$ is a $n\times n$ mass matrix, defining the mass distribution over the coordinates, $\mathbf{Q}$ is the vector representing external applied forces to the system, and $\delta\mathbf{q}^*$ is an infinitesimal displacement compatible with the constraints. The expressions for $\mathbf{M}$ and $\mathbf{Q}$ matrices are coordinate-dependent.

By definition, virtual displacements $\delta\mathbf{q}^*$ are instantaneous, and unaffected by time-dependent constraints. The Jacobian matrix of the constraints, $\mathbf{\Phi_q}$, indicates the directions for which the constraints are violated. Thus, virtual displacements $\delta\mathbf{q}^*$ are orthogonal to the Jacobian rows, yielding

$$\mathbf{\Phi_q}(\mathbf{q},t)\delta\mathbf{q}^*=\mathbf{0} \tag{2.3}$$

Ideal constraints do not produce work, therefore they do not appear in (2.1). In order to get the equations for the dynamic equilibrium, constraint forces must be introduced into this expression. The rows of $\mathbf{\Phi_q}$ give the direction of the reaction forces associated with each constraint. Each one of those rows is multiplied by a unknown λ_i value that holds the magnitude of the i^{th} force [35]:

$$\delta\mathbf{q}^{*\mathrm{T}}(\mathbf{M}\ddot{\mathbf{q}}+\mathbf{\Phi}_\mathbf{q}^\mathrm{T}\boldsymbol{\lambda}-\mathbf{Q}(\mathbf{q},\dot{\mathbf{q}}))=\mathbf{0} \tag{2.4}$$

Since there are only $n-m$ independent coordinates and m unknown λ_i values, the final system can be expressed as

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi}_\mathbf{q}^\mathrm{T}\boldsymbol{\lambda} = \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.5}$$

$$\mathbf{\Phi}(\mathbf{q}, t) = \mathbf{0} \tag{2.6}$$

This is a system of Differential-Algebraic Equation (DAE) which is not usually solved directly. Non-linear DAEs with a high index lead to very complex solving algorithms, as described in [7] and [31]. This complexity entails further disadvantages as equation instability problems, and the requirement of specific integration methods.

For multibody dynamics, this complexity can be avoided by transforming the motion equations (2.5) into a simpler to solve, equivalent system that approximates the correct solution. Furthermore, it will be shown that some of these methods lead to more compact system sizes than the original DAE system of $n + m$ equations.

2.2.1. Penalty method

The penalty method transforms (2.5) into a system of Ordinary Differential Equations (ODE), leading to simpler integration algorithms. In this method, constraint forces are considered proportional to the violation of the constraints and their derivatives, as shown in (2.7). The physical meaning of this method is that constraints are substituted by an equivalent mass-spring-damping system that tries to prevent disallowed displacements, as shown in Figure 2.1. In that figure, a simple pendulum mechanism is represented with a punctual mass at its end. If the weight of the bar is negligible, the mass can be considered free, and the penalty forces (2.7) will act as a very rigid and dissipating system that maintains constant the length of the bar.

$$\boldsymbol{\lambda} = \alpha(\ddot{\mathbf{\Phi}} + 2\xi\omega\dot{\mathbf{\Phi}} + \omega^2\mathbf{\Phi}) \tag{2.7}$$

The penalty parameter α can be usually chosen as large as 10^6 or 10^7 for many real life mechanisms. The other two parameters are used to impose an elastic and dissipating behavior into the constraint forces. Typically recommended values are $\omega = 10$ and $\xi = 1$ (critical dissipation).

Substituting (2.7) into (2.5), a linear system of equation is obtained, being $\ddot{\mathbf{q}}$ its sole unknown.

$$(\mathbf{M} + \alpha\mathbf{\Phi}_\mathbf{q}^\mathrm{T}\mathbf{\Phi}_\mathbf{q})\ddot{\mathbf{q}} = \mathbf{Q} - \alpha\mathbf{\Phi}_\mathbf{q}^\mathrm{T}(\dot{\mathbf{\Phi}}_\mathbf{q}\dot{\mathbf{q}} + 2\xi\omega\dot{\mathbf{\Phi}} + \omega^2\mathbf{\Phi}) \tag{2.8}$$

$$\ddot{\mathbf{q}} = (\mathbf{M} + \alpha\mathbf{\Phi}_\mathbf{q}^\mathrm{T}\mathbf{\Phi}_\mathbf{q})^{-\mathbf{1}}[\mathbf{Q} - \alpha\mathbf{\Phi}_\mathbf{q}^\mathrm{T}(\dot{\mathbf{\Phi}}_\mathbf{q}\dot{\mathbf{q}} + \mathbf{2}\xi\omega\dot{\mathbf{\Phi}} + \omega^2\mathbf{\Phi})] \tag{2.9}$$

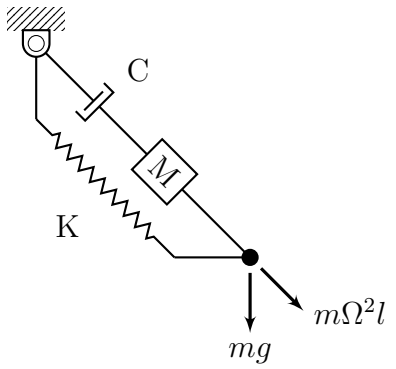

Figure 2.1: Physical meaning of penalty factors.

2.2.2. Index-3 Augmented Lagrangian

A more advanced approach is the Index-3 Augmented Lagrangian, with penalty only at position level, and mass-stiffness-damping orthogonal projections. This formulation fulfills the constraints but not their derivatives; therefore, the obtained velocities and accelerations must be further processed to make them also fulfill the derivatives of the constraints. That *cleaning* process is called projection, since it consists in the projection of $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ over the manifolds represented by $\dot{\mathbf{\Phi}}$ and $\ddot{\mathbf{\Phi}}$.

The Augmented Lagrangian method combines the penalty method, where a high α factor is penalizing constraint violations, with the Lagrange multipliers, which hold the magnitude of the reaction forces that avoid the violation of the constraints. Unlike the Lagrangian method, the system size is $n \times n$ —being n the number of coordinates—, since this time the Lagrange multipliers are computed in an iterative manner, and the constraints are not solved explicitly:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\lambda}^* + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\mathbf{\Phi} = \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.10}$$

$$\boldsymbol{\lambda}_{i+1}^* = \boldsymbol{\lambda}_i^* + \alpha\mathbf{\Phi}_{i+1},\ i = 0, 1, 2, \ldots \tag{2.11}$$

As mentioned, Lagrange multipliers $\boldsymbol{\lambda}_i^*$ are computed according to the current state of the system, and they will converge until they reach equilibrium values. At the first iteration of the initial time-step, the multipliers $\boldsymbol{\lambda}_0^*|_{t=0}$ can be computed by means of the penalty method described in (2.7). For the rest of the time-steps, the final value of the multipliers in the previous instant, $\boldsymbol{\lambda}_n^*|_t$, is used for the first iteration, $\boldsymbol{\lambda}_0^*|_{t+h}$.

2.3. Integration of the equations of motion

2.3.1. Newmark integrators

This family of integrators are specific methods for solving second order differential equation systems. This technique has been widely used in multibody dynamics. Among structural integrator families, Newmark [43], HHT [32] and Generalized-α [9] are the most common ones. The expressions for these implicit integrators can be used to compute positions and velocities from the coordinate values and its derivatives of the previous time-step and the accelerations in the current time-step. For example, the Newmark integrator family,

$$\mathbf{q}_{n+1} = \mathbf{q}_n + h\dot{\mathbf{q}}_n + \frac{h^2}{2}\{(1-2\beta)\ddot{\mathbf{q}}_n + 2\beta\ddot{\mathbf{q}}_{n+1}\} \tag{2.12}$$

$$\dot{\mathbf{q}}_{n+1} = \dot{\mathbf{q}}_n + h\{(1-\gamma)\ddot{\mathbf{q}}_n + \gamma\ddot{\mathbf{q}}_{n+1}\} \tag{2.13}$$

From 2.12 and 2.13 velocities and accelerations in time-step n+1 can be expressed as function of positions in time-step n+1 (primary variables).

$$\dot{\mathbf{q}}_{n+1} = \frac{\gamma}{\beta h}\mathbf{q}_{n+1} + \hat{\dot{\mathbf{q}}}_n; \quad \hat{\dot{\mathbf{q}}}_n = -\left[\frac{\gamma}{\beta h}\mathbf{q}_n + \left(\frac{\gamma}{\beta} - 1\right)\dot{\mathbf{q}}_n + \left(\frac{\gamma}{2\beta} - 1\right)\ddot{\mathbf{q}}_n\right] \tag{2.14}$$

$$\ddot{\mathbf{q}}_{n+1} = \frac{1}{\beta h^2}\mathbf{q}_{n+1} + \hat{\ddot{\mathbf{q}}}_n; \quad \hat{\ddot{\mathbf{q}}}_n = -\left[\frac{1}{\beta h^2}\mathbf{q}_n + \frac{1}{\beta}h\dot{\mathbf{q}}_n + \left(\frac{1}{2\beta} - 1\right)\ddot{\mathbf{q}}_n\right] \tag{2.15}$$

Introducing now 2.14 and 2.15 into (2.10), a non-linear system of equations is obtained in which the positions in time-step n+1 are the unknowns. A factor of $h^2/4$ is used in order to avoid having to scale the mass matrix:

$$f(\mathbf{q}) = \mathbf{M}\mathbf{q}_{n+1} + \frac{h^2}{4}\mathbf{\Phi}^{\mathrm{T}}_{\mathbf{q}_{n+1}}(\alpha\mathbf{\Phi}_{n+1} + \boldsymbol{\lambda}_{n+1}) - \frac{h^2}{4}\mathbf{Q}_{n+1} + \frac{h^2}{4}\mathbf{M}\hat{\ddot{\mathbf{q}}}_n = \mathbf{0} \tag{2.16}$$

This system can be solved using well-known methods such as the Newton-Raphson iterative solver. The generic method is presented as

$$\left[\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\right]_i \Delta\mathbf{q}_{i+1} = -[f(\mathbf{q})]_i \tag{2.17}$$

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \Delta\mathbf{q}_{i+1} \tag{2.18}$$

For this problem, the Jacobian of (2.16) with respect to the coordinates $\mathbf{q}$ is:

$$\left[\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\right] = \mathbf{M} + \frac{h^2}{4}\left\{\mathbf{\Phi_{qq}}^{\mathrm{T}}(\alpha\mathbf{\Phi} + \boldsymbol{\lambda}) + \mathbf{\Phi_q}^{\mathrm{T}}(\alpha\mathbf{\Phi_q} + \boldsymbol{\lambda}_\mathbf{q}) + \mathbf{K} + \frac{2}{h}\mathbf{C}\right\} \tag{2.19}$$

The expression (2.19) is denoted as the *tangent matrix*. The computation of some elements of this matrix can be avoided, given their negligible magnitude with respect to the remaining matrix terms. $\mathbf{\Phi}^{\mathrm{T}}_{\mathbf{qq}}$ is a very sparse third order tensor vastly composed of null values; the rest of its elements are usually constant, since the constraints are generally linear or at most quadratic when using natural coordinates. The computation of the combined term $\mathbf{\Phi}^{\mathrm{T}}_{\mathbf{qq}}(\alpha\mathbf{\Phi}+\boldsymbol{\lambda})$ can be avoided, both for simplification and for speed up purposes. A remark should be made about the fact that it is not strictly required to compute the exact tangent to achieve a convergent method.

Therefore, an approximation for the *tangent matrix* with good convergence properties is

$$\left[\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\right] \approx \mathbf{M} + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}(\mathbf{\Phi}^{\mathrm{T}}_{\mathbf{q}}\alpha\mathbf{\Phi_q} + \mathbf{K}) \tag{2.20}$$

being

$$\mathbf{K} = -\frac{\partial \mathbf{Q}}{\partial \mathbf{q}} \tag{2.21}$$

$$\mathbf{C} = -\frac{\partial \mathbf{Q}}{\partial \dot{\mathbf{q}}} \tag{2.22}$$

The $\mathbf{K}$ and $\mathbf{C}$ terms are called the *stiffness* and *damping* matrices. They state the influence of the change in positions and velocities on the magnitude of the applied forces. Those parameters also resemble the stiffness and damping in the well-known *harmonic oscillator* problem, shown in Figure 2.2. In the *harmonic oscillator*, the applied force vector is $F = kx + c\dot{x}$. Constant k is the ratio at which the force increases given an increase in the displacement x, that is $\partial F/\partial x = k$. Constant c behaves in the same manner with respect to the displacement time derivative, $\partial F/\partial \dot{x} = c$. Therefore, the naming for the expressions (2.21) and (2.22) is apparent.

The residual for the Newton-Raphson iterative method is computed directly for the step $n+1$ from equation (2.16):

$$f(\mathbf{q}) = \frac{h^2}{4}(\mathbf{M}\ddot{\mathbf{q}} + \mathbf{\Phi}^{\mathrm{T}}_{\mathbf{q}}\boldsymbol{\lambda}^* + \mathbf{\Phi}^{\mathrm{T}}_{\mathbf{q}}\alpha\mathbf{\Phi} - \mathbf{Q})_{n+1} \tag{2.23}$$

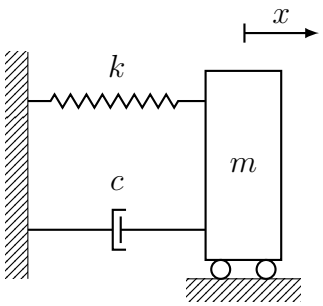

Figure 2.2: Harmonic oscillator

As noted in [10], the approximate tangent matrix (2.20) can be suffer from loss of significance problems when the integration time step, h, is very small. The exact value depends on the order of magnitude of the parameters of the problem (masses, forces, etc.), and can be as small as $10^{-6}s$ for typical, common problems.

2.3.2. Projections of velocities and accelerations

Since the Index-3 Augmented Lagrangian method has the position coordinates $\mathbf{q}$ as its primary variables, the computed solutions do satisfy the imposed constraints, $\mathbf{\Phi} = \mathbf{0}$, at the requested precision level. However, the equations presented so far do not impose the fulfillment of the constraints' derivatives, $\dot{\mathbf{\Phi}} = \mathbf{0}$ and $\ddot{\mathbf{\Phi}} = \mathbf{0}$. In order to avoid instabilities that could arise when integrating incoherent sets of velocities and accelerations, a projection process can be performed for keeping the velocities and accelerations on the constraints derivatives manifolds.

The projection method is based in the work developed in [1], which described the projection of velocities and accelerations onto the mass matrix. The original formulation is

$$\begin{aligned} \min V &= \frac{1}{2}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{M}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) \\ &\text{subject to } \dot{\mathbf{\Phi}}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{0} \end{aligned}$$

This problem is solved using the Augmented Lagrange Multipliers method described before,

$$\min V^* = \frac{1}{2}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{M}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) + \frac{1}{2}\dot{\mathbf{\Phi}}^{\mathrm{T}}\alpha\dot{\mathbf{\Phi}} + \dot{\mathbf{\Phi}}^{\mathrm{T}}\boldsymbol{\sigma} \tag{2.24}$$

where $\boldsymbol{\sigma}$ is the vector of multipliers for the projection problem. Then,

$$\frac{\partial V^*}{\partial \dot{\mathbf{q}}} = \mathbf{M}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) + \dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}}^{\mathrm{T}}\alpha\dot{\mathbf{\Phi}} + \dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}}^{\mathrm{T}}\boldsymbol{\sigma} = \mathbf{M}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\dot{\mathbf{\Phi}} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\sigma} = 0 \tag{2.25}$$

The identity $\dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}} = \mathbf{\Phi}_{\mathbf{q}}$ is demonstrated with the expression (2.26)

$$\dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}} = \frac{\partial \dot{\mathbf{\Phi}}}{\partial \dot{\mathbf{q}}} = \frac{\partial}{\partial \dot{\mathbf{q}}}(\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} + \mathbf{\Phi}_t) = \mathbf{\Phi}_{\mathbf{q}} \tag{2.26}$$

This projection method is also an iterative process where a better, new set of velocities, $\dot{\mathbf{q}}$, will be obtained from the original velocities, $\dot{\mathbf{q}}^*$, coming from the integrator. The multipliers $\boldsymbol{\sigma}$ are updated as

$$\boldsymbol{\sigma}_{i+1} = \boldsymbol{\sigma}_i + \alpha \dot{\mathbf{\Phi}}_{i+1} \tag{2.27}$$

An interesting computing performance improvement is to use the penalty optimization method instead of the Augmented Lagrangian method. The former problem is not iterative, leading instead to a linear system of equations:

$$(\mathbf{M} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha \mathbf{\Phi}_{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{M}\dot{\mathbf{q}}^* - \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha \mathbf{\Phi}_t \tag{2.28}$$

Additionally, it is a sufficient condition for finding the minimum that $\mathbf{M} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha \mathbf{\Phi}_{\mathbf{q}}$ matrix is positive definite [45].

The projection method used in this document applies some additional optimizations. A first performance improvement comes from the strategy of avoiding to compute and factorize the coefficient matrix of the linear system, $\mathbf{M} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha \mathbf{\Phi}_{\mathbf{q}}$. Cuadrado et al. [11] demonstrated that it is also possible to attain the projection of velocities using the tangent matrix (2.20) instead. This matrix was already computed and factorized in previous steps when solving the dynamics. Thus, choosing a new definition for the projection problem such as

$$\min V = \frac{1}{2}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)^{\mathrm{T}} \mathbf{P} (\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) \tag{2.29}$$

$$\text{subject to } \frac{h^2}{4}\dot{\mathbf{\Phi}}(\mathbf{q}, \dot{\mathbf{q}}, t) = \mathbf{0} \tag{2.30}$$

where $\mathbf{P}$ is a matrix defined as

$$\mathbf{P} = \mathbf{M} + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}\mathbf{K} \tag{2.31}$$

The penalty method leads to the minimization problem

$$\min V^* = \frac{1}{2}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)^{\mathrm{T}} \mathbf{P} (\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) + \frac{h^2}{8}\dot{\mathbf{\Phi}}^{\mathrm{T}} \alpha \dot{\mathbf{\Phi}} + \dot{\mathbf{\Phi}}^{\mathrm{T}} \boldsymbol{\sigma} \tag{2.32}$$

Differentiating and solving the equation

$$\frac{\partial V^*}{\partial \dot{\mathbf{q}}} = \mathbf{P}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*) + \frac{h^2}{4}\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha \dot{\mathbf{\Phi}} = 0 \tag{2.33}$$

the following linear system is found:

$$(\mathbf{P} + \frac{h^2}{4}\mathbf{\Phi_q}^{\mathrm{T}}\alpha\mathbf{\Phi_q})\dot{\mathbf{q}} = \mathbf{P}\dot{\mathbf{q}}^* - \frac{h^2}{4}\mathbf{\Phi_q}^{\mathrm{T}}\alpha\mathbf{\Phi}_t \tag{2.34}$$

Substituting the weight matrix $\boldsymbol{P}$ with its own value, the final expression is

$$(\mathbf{M} + \frac{h}{2}\mathbf{C} + \frac{\mathbf{h^2}}{\mathbf{4}}(\mathbf{\Phi_q}^{\mathrm{T}}\alpha\mathbf{\Phi_q} + \mathbf{K}))\dot{\mathbf{q}} = (\mathbf{M} + \frac{\mathbf{h}}{\mathbf{2}}\mathbf{C} + \frac{\mathbf{h^2}}{\mathbf{4}}\mathbf{K})\dot{\mathbf{q}}^* - \frac{\mathbf{h^2}}{\mathbf{4}}\mathbf{\Phi_q}^{\mathrm{T}}\alpha\mathbf{\Phi_t} \tag{2.35}$$

The coefficient matrix of this final system is the *tangent matrix* (2.20) already computed and factorized in the last iteration of the motion problem (2.17). Coincidentally, the *tangent matrix* is definite-positive as well; this fact ensures that the algorithm finds the minimum of the optimization problem (2.30).

The accelerations' projection method is analogous to the velocities' projection already presented. The coordinates $\ddot{\boldsymbol{q}}$ are projected onto a matrix and required to fulfill the constraints. For the mass matrix-orthogonal projection:

$$\begin{aligned} \min \mathrm{V} &= \frac{1}{2}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{M}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*) \\ \text{subject to } &\ddot{\mathbf{\Phi}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) = 0 \end{aligned}$$

And for the corresponding Augmented Lagrangian method:

$$\min \mathrm{V}^* = \frac{1}{2}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{M}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*) + \frac{1}{2}\ddot{\mathbf{\Phi}}^{\mathrm{T}}\alpha\ddot{\mathbf{\Phi}} + \ddot{\mathbf{\Phi}}^{\mathrm{T}}\boldsymbol{\nu} \tag{2.36}$$

Here $\boldsymbol{\nu}$ are the Lagrange multipliers for the problem of the acceleration projection. The expression resulting from differentiating this equation in order to find its minimum:

$$\begin{aligned} \frac{\partial V^*}{\partial \ddot{\mathbf{q}}} =& \mathbf{M}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*) + \ddot{\mathbf{\Phi}}_{\ddot{\mathbf{q}}}^{\mathrm{T}}\alpha\ddot{\mathbf{\Phi}} + \ddot{\mathbf{\Phi}}_{\ddot{\mathbf{q}}}^{\mathrm{T}}\boldsymbol{\nu} \\ =& \mathbf{M}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*) + \mathbf{\Phi_q}^{\mathrm{T}}\alpha(\mathbf{\Phi_q}\ddot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_t) + \mathbf{\Phi_q}^{\mathrm{T}}\boldsymbol{\nu} = 0 \end{aligned} \tag{2.37}$$

Where the property $\ddot{\mathbf{\Phi}}_{\ddot{\mathbf{q}}} = \frac{\partial \ddot{\mathbf{\Phi}}}{\partial \ddot{\mathbf{q}}} = \frac{\partial}{\partial \ddot{\mathbf{q}}}(\Phi_{\mathbf{q}}\ddot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_t) = \mathbf{\Phi_q}$ has been applied.

In the same way as the velocity projection, a simpler problem can be solved using only a penalty method. Therefore, a linear system of equations is obtained:

$$(\mathbf{M} + \mathbf{\Phi_q}^{\mathrm{T}}\alpha\mathbf{\Phi_q})\ddot{\mathbf{q}} = \mathbf{M}\ddot{\mathbf{q}}^* - \mathbf{\Phi_q}^{\mathrm{T}}\alpha(\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_t) \tag{2.38}$$

The *tangent matrix* can be used as well in the acceleration case by redefining the problem in terms of the $\boldsymbol{P}$ matrix already presented. The formulation of the problem would be:

$$\min V = \frac{1}{2}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{P}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*) \tag{2.39}$$

$$\text{subject to } \frac{h^2}{4}\ddot{\mathbf{\Phi}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) = 0 \tag{2.40}$$

and the final linear equation system:

$$(\mathbf{M}\frac{h}{2}\mathbf{C} + \frac{h^2}{4}(\Phi_{\mathbf{q}}^{\mathrm{T}}\alpha\mathbf{\Phi}_{\mathbf{q}} + \mathbf{K}))\ddot{\mathbf{q}} = (\mathbf{M} + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}\mathbf{K})\ddot{\mathbf{q}}^* - \frac{h^2}{4}\Phi_{\mathbf{q}}^{\mathrm{T}}\alpha(\dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_t) \tag{2.41}$$

2.4. Flowchart

The flowchart for the Augmented Lagrangian method is displayed in Figure 2.3.

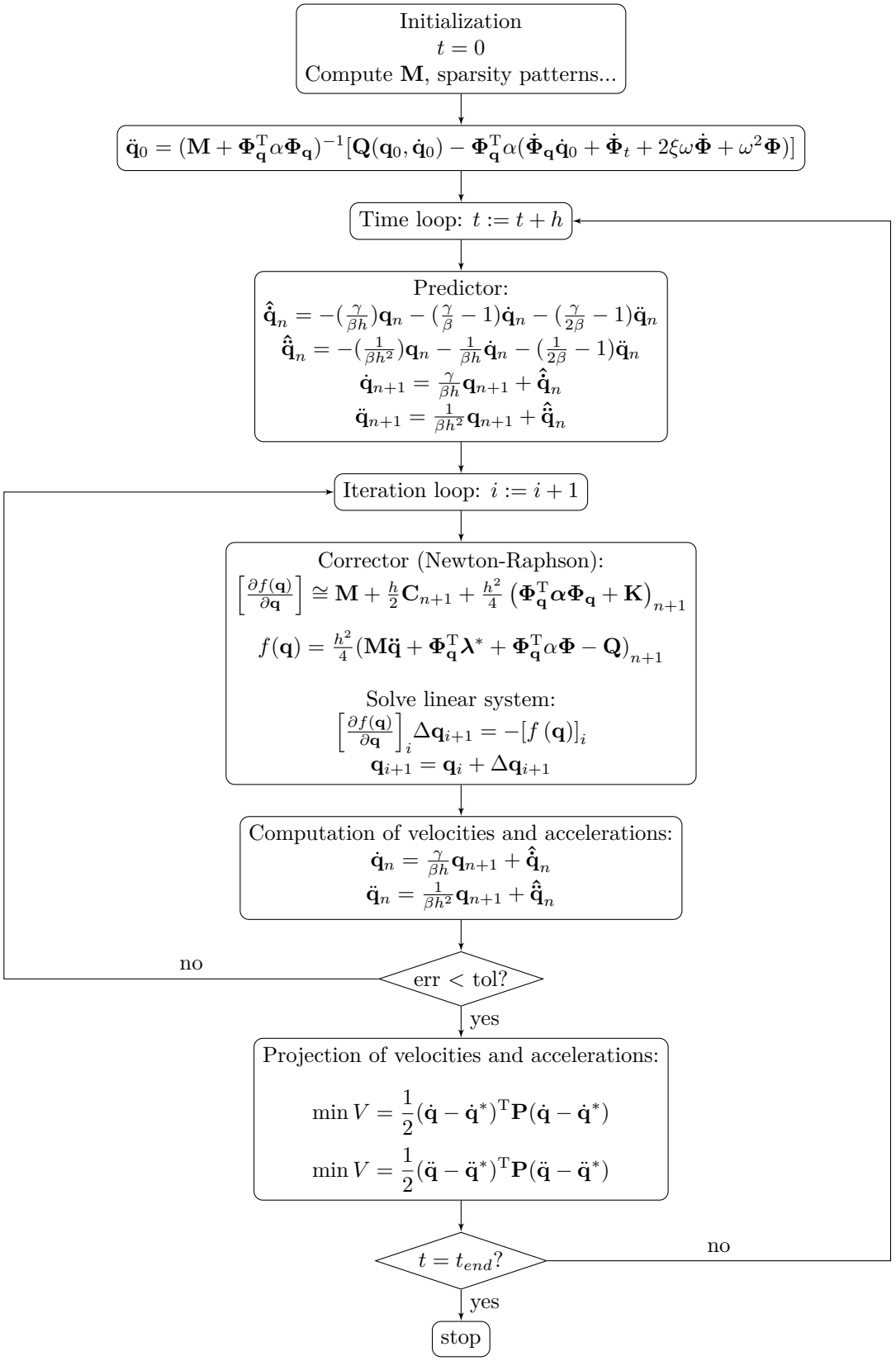

Figure 2.3: Flowchart for the Augmented Lagrangian method.

Chapter 3

Contact model

Collisions and contacts are fundamental phenomena in almost every complex multibody simulation. While some physics simulations may avoid the need to handle impacts, most of them, including machinery, vehicles or virtual assembly have to deal with this kind of events. There is no general consensus about which is the most appropriate way of implementing a contact model. In the framework of this work, the model is composed of the contact detection algorithm and the contact force model itself. The contact detection algorithms can vary between trivial contact detections to very complex general algorithms like the one proposed here. Force models can largely vary from one simulator to another, each one must match different needs and have distinct goals and sometimes different approaches are possible to solve the same problem. While some simulations may do well with simplified force models (like in those cases where only very primitive contacts take place or where precision is not a main concern), others may need more accurate models to characterize more complex contact situations.

Furthermore, the decision to choose among all the diverse models available is constrained by the CPU time during the simulation. Regarding this issue, we can catalog all simulations in two big different groups: those where the simulation is run completely offline, so the CPU time is not a priority, and those where the user or other entities (like sensors or actuators) somehow interact with the simulation. In this last category we find hardware-in-the-loop and human-in-the-loop simulators, which need real time execution in order to provide interactive interfaces. For this ones, the force evaluation time is critical, in the sense that only models which are capable of computing forces much faster than the time-step are suitable since the overall calculations have to be done in less time than the time-step.

A worst case scenario would be a simulation where both real time and a high degree of precision are needed, since this represents an execution time constraint and a need for complex —and therefore computer time demanding— force models. This would be the case for virtual assembly simulations, where the user requires

that the visual feedback matches the speed of the decision-making loop but also that the interaction between virtual bodies is smooth enough to achieve things like handling and assembling small parts together.

The process of computing contact forces is divided into two stages: the collision detection, where bodies in a potential contact situation are identified and the interference properties are calculated, and the computation of the forces that will be responsible for generating a motion that resembles the real one.

During this chapter, the force models selected for this thesis are presented. Usually, two different contact force models are needed to characterize object interference: one that accounts for the normal force, and another one to calculate the tangential force. The normal force is needed to avoid object inter-penetration, while the tangential force illustrates the friction between bodies.

In this work, triangular meshes are used as the object representation, and only rigid bodies with small local deformations in the contacting regions are considered, so no object deformation is taken into account in terms of the geometry representing the body.

3.1. Normal contact

3.1.1. Description

During the development of this work, a general contact model capable of dealing with multiple contacts and conforming situations was pursued. As explained in Section 1.2.2, many continuous force models are based on the Hertz theory and thus assume contact areas much smaller than the characteristic dimensions of the contacting bodies, i.e., non-conforming contacts. In mechanism and machinery simulation it is very common to find conforming contacts that fall out of this assumption, such as plane parts resting on flat surfaces, or a pin inside a cylinder.

For this reason, the force model selected for the normal force computation was the Gonthier volumetric model described in [26]. It is based on a modified Winkler elastic foundation model and, in contrast with other models more suited to point contacts, it mimics the force distribution that takes place between objects in contact due to the body deformation. Furthermore, the fact of having a volume of interference instead of a surface allows for the calculation of properties that are useful to include rolling resistance and spinning friction in the model.

Since these volumetric properties can always be calculated for any intersection, the method is not restricted to contacts over a small area compared with the bodies size: it can be considered a generalized model for any kind of situation.

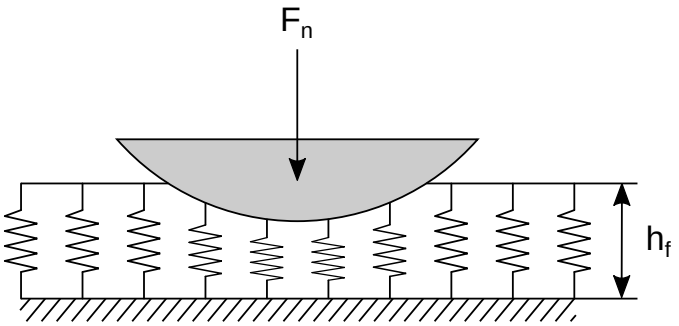

Figure 3.1: Winkler elastic foundation

The expression for the normal force is the following:

$$\mathbf{F_n} = \frac{k_n}{h_n} V(1 + a v_n)\mathbf{n} \tag{3.1}$$

where k_n is the contact stiffness, h_n the hysteretic damping factor, V the volume of the intersection, a is the hysteretic damping factor [27] and v_n stands for the relative normal velocity of both bodies in contact. n is the direction of the force and it is also used in some subscripts standing for "normal".

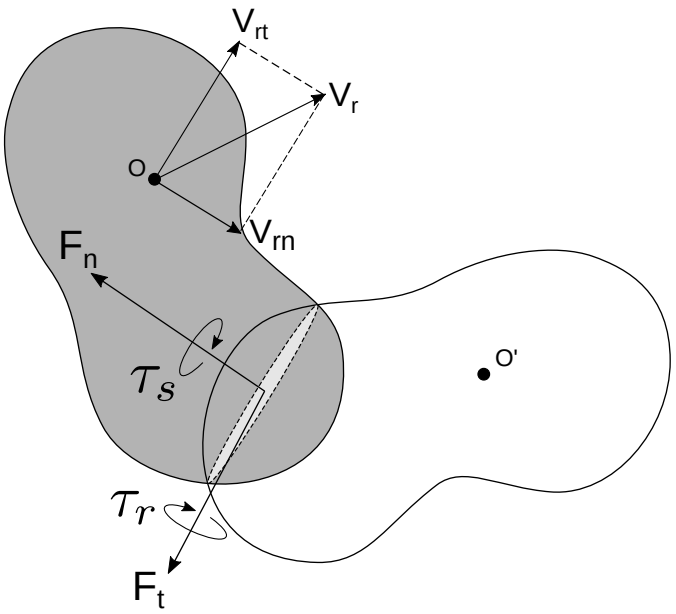

Figure 3.2: Contact forces and momenta during a collision

3.2. Tangential contact

The tangential model is composed of three different sub-models that are suited for different friction situations: sliding/sticking, rolling and spinning. Each of these models actuate only in their corresponding situations as they are proportional to different velocities involved in friction phenomena. Sliding and sticking is related to the tangential velocity and generates an associated force. Rolling is related to the tangential angular velocity and spinning to the normal angular velocity, and both of them are associated with a resistive momentum.

3.2.1. Sliding friction and stiction

The Gonthier volumetric force [26] proposes the following sliding force model

$$\mathbf{F_t} = F_n(\mathbf{v_t} + \boldsymbol{\omega_t} \times \boldsymbol{\rho_n}) \tag{3.2}$$

where F_n stands for the normal force, $\mathbf{v_t}$ is the relative tangential velocity, $\boldsymbol{\omega_t}$ is the angular velocity and $\boldsymbol{\rho_n}$ the position vector of the contact surface centroid relative to the interference volume centroid.

The paper also specifies that, for rigid bodies, $\boldsymbol{\rho_n}$ can be neglected, simplifying the expression to

$$\mathbf{F_t} = F_n\mathbf{v_t} \tag{3.3}$$

The problem with this approach is the lack of static friction. If this model is used to simulate a body sliding down a slope, the body will be never be stopped completely by the friction. As we can see in 3.3, the tangential force is directly proportional to the tangential velocity. For this reason, when the velocity of the body tends to zero, its small tangential velocity will yield a tiny tangential force, thus not being able to stop the object and letting it slide at a very small velocity [21]. To solve this limitation, the model described in [17] was implemented.

$$\mathbf{F_t} = \kappa\mathbf{F_{stick}} + (1-\kappa)\mathbf{F_{slide}} - \mu_{visc}\mathbf{v_t} \tag{3.4}$$

Here, three different terms are depicted. The first one accounts for the static friction, introducing a damper-spring force that acts at very low velocities to solve the null velocity problem described above. This force can be considered as the result of small viscoelastic elements that actuate between quasi-static colliding bodies and are known as bristles. The second term represents the dynamic friction (Coulomb model), and the last one constitutes the viscous friction, but in this work μ_{visc} was set to zero as a simplification. The smoothing function κ is responsible for

the transition between null or low tangential velocities (stick) and normal dynamic conditions (slip) and must be selected so that it fulfills the following conditions:

$$\kappa = \left\{ \begin{array}{ll} 0; & \|\mathbf{v}_t\| >> v_{stick} \\ 1; & \|\mathbf{v}_t\| = 0 \end{array} \right\} \tag{3.5}$$

where v_{stick} is the transition parameter and can be understood as the velocity where the stick phase ends. The chosen form for this function, as described in [26], is:

$$\kappa = \mathrm{e}^{-\left(\mathbf{v}_t^{\mathrm{T}}\mathbf{v}_t\right)/v_{stick}^2} \tag{3.6}$$

3.2.2. Rolling resistance

The previous friction model works well for simple situations where a body slides over another body, but does not account for the friction generated when there exists rolling. In this case, the contact patch in the moving bodies has near-zero velocity and the stiction bristles are rendered useless since the hooking point varies constantly. Gonthier proposes the following expression to calculate the rolling resistance

$$\boldsymbol{\tau_r} = \frac{k_n a}{h_n}\mathbf{I_g}\boldsymbol{\omega}_{\mathrm{t}} \tag{3.7}$$

where $\mathbf{I_g}$ is the intersection volume inertia tensor and $\boldsymbol{\omega}_{\mathrm{t}}$ the tangential angular velocity. This expression represents a torque opposing to the relative tangential angular velocity.

3.2.3. Spinning friction

There is another friction that must be taken into account: the one derived form spinning bodies. Gonthier also presented a model for this situations.

$$\boldsymbol{\tau_s} = \frac{F_n}{V}\mathbf{I_g}\boldsymbol{\omega}_{\mathrm{n}} \tag{3.8}$$

where $\boldsymbol{\omega}_{\mathrm{n}}$ is the normal angular velocity. This equation represents a torque acting in opposition to the relative normal angular velocity.

Chapter 4

Collision detection

It is clear from Chapter 3 that, in order to be able to use the force models presented there, several pieces of information from the simulation have to be collected. Some of them are provided by the multibody system itself, as body positions and velocities. Nevertheless, there the rest of the properties go beyond the mechanical definition of a rigid body solid, that deals with mass distribution, since they explicitly depend on the geometrical definition of the boundaries.

The model described in Section 3.1 requires that the properties of the volumes defined by the intersection of each pair of colliding bodies are known. In order to do that, it must be ensured that: *a*) those properties can be computed and; *b*) this is done with a reasonable degree of precision at a high computation rate.

Those tasks are highly-dependent on the chosen geometric representation method for the surface of the bodies, but when considered from a high-level view, they can be always divided in the following sub-steps:

1. Detect pairs of bodies in collision subject to reaction forces.

2. Trim the surfaces that define the inter-penetration.

3. Obtain the desired parameters out of those surfaces: penetration length or volume, perpendicular direction of the contact.

In this chapter, those stages will be described for the two geometric method proposed: definition by triangular meshes or by a distribution of inner spheres, the Inner Sphere Tree (IST).

4.1. Detection and characterization of contacts between solids

4.1.1. Geometric definition of surfaces

The development of a virtual assembly simulator comprises managing shapes that are arbitrarily defined. Mechanical parts are very frequently designed with the aid of Computer Aided Design (CAD) software, allowing defining complex parts by de-composing them into sequences of elemental shapes and geometric procedures. The results are stored as a set of free-form surfaces that can define any part as a solid domain. This encoding is called Boundary Representation (B-Rep), since all the properties of the solid are inferred from the surfaces that enclose it. Those closed solid domains can be later used into any other engineering procedure, since they can hold all the physical information of the part. Popular uses of CAD geometry are the FEM for stress-strain computation, impact, thermal distribution, Computer Fluid Dynamics (CFD), etc.

Representing a solid in terms of the bounding surfaces that enclose its volume is not the only form of defining a 3D part. Another interesting method is the Constructive Solid Geometry (CSG), in which a part is defined in terms of boolean operations between simple primitives. Those primitives are shapes for which usually a mathematical definition of their surface can be found: spheres, cones, cylinders... The resulting sequence of boolean operations also leads to straightforward collision detection among objects; however, although many mechanical parts are well-suited to be encoded by this method, some others pose difficult or impossible problems when modeling.

Non-Uniform Rational B-Spline (NURBS) are parametric surfaces, so each of them is defined in terms of two parameters, s and t (4.1). A direct consequence of this fact is that the process of finding features regarding one or more NURBS surfaces lead to non-linear systems whose solution time cannot be guaranteed for real-time purposes. Therefore, a common approach is to approximate the surfaces by a collection of much simpler primitives, which can be queried in a much straightforward and efficient way [41].

$$\mathbf{p}(s,t) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} h_{ij}\mathbf{p}_{ij}N_{i,K}(s)N_{j,L}(t)}{\sum_{i=0}^{n}\sum_{j=0}^{m} h_{ij}N_{i,K}(s)N_{j,L}(t)} \quad 0 \leq s,t \leq 1 \tag{4.1}$$

In the context of this document, the CAD representation is used to compute the physical properties of the contact state of two solids in collision. That information can be computed from the B-Rep definition of the objects and their relative position. However, the B-Rep is not well suited for real-time purposes. Parametric surfaces are highly flexible, but it is precisely this flexibility what vastly

complicates the surface-surface intersection evaluation in the shape of a nonlinear system. Solving a nonlinear system is a complex process in numerical analysis, and despite there are specialized textbooks on the topic, geometric modeling applications pose severe robustness, accuracy, automation, and efficiency requirements on those solvers [49]. Furthermore, the solution of a surface-surface intersection may be empty, or include a several branches curve, a surface patch or a point (Figure 4.1).

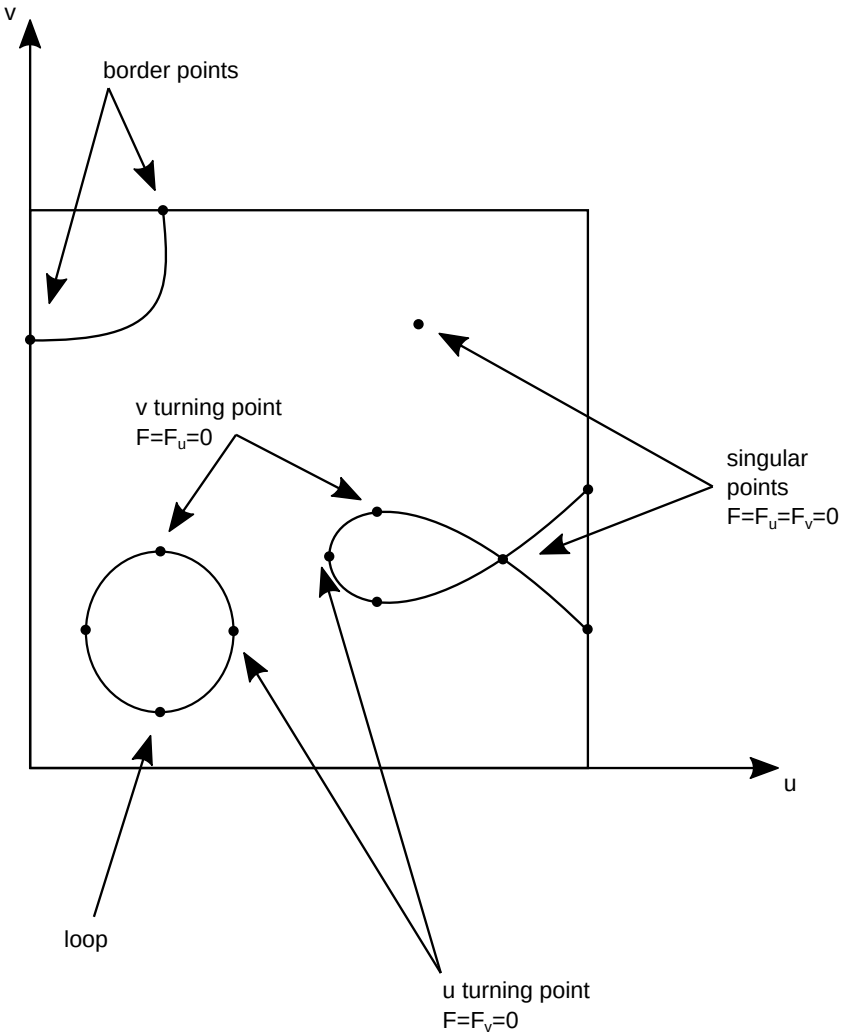

Figure 4.1: Parameter space of $\mathbf{r}(u, v)$ and resulting algebraic curve $F(u, v) = 0$ [49]

Therefore, usually the NURBS B-Rep definition is further processed in order to obtain a simpler form, easier to manage in the context of a real-time simulation.

For the end user this does not constitute any disadvantage, since the preprocessing of the shapes can be run automatically, without further intervention.

There are many ways in which the solid information can be simplified. They usually involve some kind of discretization into simple elements or primitives. In this document, two of those methods are shown: one consists in the discretization of the surface of the solid into surface primitives, and the other de-composes the volume of any part into volume primitives.

The first approach described in this document is to segment the NURBS surfaces into a collection of simple polygons —triangles— that are faster and easier to manage. Triangle polygons feature some desirable properties, as being easy to define and store in memory, and they always define a planar-subdomain independently of the value of their coordinates. On the other hand, they are not immune to bad numerical conditioning: if the length of any of its edges is much shorter than the others, or if the area of the whole triangle itself is small, any of the computations based on their data will expose a significant degree of error.

This problem shows up in several other fields that use element-based meshes for computation purposes. An example is the FEM, in which the solids are also modeled by means of surface or volume elements. Therefore, a critical aspect for obtaining good results in a FEM simulation is to start from a well-defined mesh. Usually FEM packages are able to generate good quality meshes by reaching segmentation solutions in which all its elements have a maximum level of degeneracy. We use those mesh programs to obtain surface discretizations that do not lead to numerical errors when computing the intersection properties.

Even in the case that the algorithms use well-conditioned meshes, numerical errors do still occur. This is an inherent problem stemming from the use of finite-precision algebra in computers. In the most extreme cases, rounding of floating point numbers can lead to inconsistencies that lead to wrong results.

4.2. Mesh surface trimming

The procedure of trimming two contacting closed surfaces representing a pair of solids can be regarded as a specific case of a boolean operation: the intersection result between both of them.

There is extensive research about computing boolean operations among solids. Boolean operations are frequently used as modeling operations for creating more complex shapes. This is the basis of the CSG method mentioned earlier, although the latter usually only holds the information in an implicit form, since the final surfaces are not computed directly.

4.2.1. Consistency enforcing and floating-point error mitigation

Performing boolean operations on meshes to obtain the surfaces of the resulting solid implies trimming or clipping the original ones. When the surfaces are defined as a mesh of polygons, it has been demonstrated that the queries needed to perform any operation over a mesh can be based on elementary operations called *predicates*. Geometrical predicates are used to check if a point is aligned to a segment, or if it is under or below a given plane. Those checks have to be performed in order to decide if a polygon or some part of it is going to be trimmed out of the resulting boolean shape.

Not only they are useful in terms of modularization of the implementation code, but they are also used to control frequent computation errors that arise in boolean operations. Small errors when computing those predicates lead to inconsistencies that, in turn, result in inconsistent output meshes. Inconsistent meshes do not describe valid solids since they exhibit open cracks, or non-manifold surfaces that ruin the computations of the properties needed for the contact model.

The literature on boolean operations deals primarily with this problem: some of them aim to enforce the consistency of the system [56, 20], while others focus on attaining perfect rounding in floating point computations [53].

Algorithms enforcing the consistency of the results impose a set of additional rules over the predicates so the inconsistencies cannot occur. For example, Sugihara [56] discovers that the vertices of a mesh are a sub-product of the intersection of the facets or polygons that define the mesh, themselves being subject to rounding issues. Therefore, four points belonging to a same plane but two different facets could not verify the predicate of being co-planar. The inconsistencies can be cleared if each vertex $\mathbf{v}_i$ is defined in terms of the intersection of three planes π_i, with $i = 1, 2, 3$. For example, the predicate that computes if a point $\mathbf{p} = [x, y, z]^T$ is below or above a given plane in implicit form, $\pi = ax + by + cz + d$, is modified into

$$F = \begin{vmatrix} a_1 & b_1 & c_1 & d_1 \\ a_2 & b_2 & c_2 & d_2 \\ a_3 & b_3 & c_3 & d_3 \\ a_\pi & b_\pi & c_\pi & d_\pi \end{vmatrix} \tag{4.2}$$

where the first three row parameters are the coefficients of the planes whose intersection is the point $\mathbf{p}$, while the last row are the parameters of the plane π. If the determinant F is positive, the point $\mathbf{p}$ is above the plane π. If it is negative, the point is below, and if the computation is equal to zero, then the point and the plane are co-planar. This re-formulation of the predicates is also immune to the

numerical errors that happen when any of the mesh operands is transformed by any affine operation as a translation, rotation or scale.

Sugihara also removes the rounding error from the predicates scaling each parameter by a sufficiently large coefficient, so all the determinants can be carried with integer number data types. The disadvantage of this method is that the requirements of storage for any mesh increase by a factor of 4. This is not a big problem in terms of storage in the current computing platforms, but it can affect greatly to its storage locality, and therefore to the performance of the memory reading bandwidth.

Another possibility for avoiding floating-point rounding and therefore model inconsistencies, is to use unlimited-precision floating point number data types. Even the computation results would be correct, the downside of this method is that neither the time nor the amount of memory spent on them can be easily predicted beforehand, so their utility in real-time applications is limited. In the development of a mesh triangulator for an earthquake FEM, Shewchuk [53] develops a progressive, adaptive system in which the predicates are split into several stages with an increasing degree of accuracy. If the accuracy at any intermediate step is enough to verify the output of predicate (usually the sign of the operation, instead of the value), the rest of the stages can be discarded, therefore saving time and computational resources. The system is based on special functions that perform arithmetic operations, and upon them, the predicates are built. Statistically, the most cases that the predicates will be used for will be clear cases in which a solution is found at the first stages, and only in the minority of the cases the difficulty of the problem will make that the algorithm has to carry all the steps.

The library of predicates is licensed as "Public Domain" and available at `https://www.cs.cmu.edu/%7Equake/robust.html`. A disadvantage of this library is that it relies in a complete IEEE-conforming floating-point computing platform; otherwise the results may not be correct. Even if the library was correct at the time it was written, there is no source which can confirm if it is still the case on nowadays computers, which feature vectorization instructions as Streaming SIMD Extensions (SSE) or Advanced Vector eXtensions (AVX). Under the most aggressive levels of code optimization of current compilers, numerical code can be wildly transformed, leading to inconsistencies when dealing with brittle algorithms.

More recently, Devillers [12] describes a system for computing the geometric predicates exactly and efficiently based on transforming their source (user provided) code into an analogous multi-step filter. As opposed to Shewchuk's approach, in which each predicate was decomposed into several stages in the same system, Devillers uses a hybrid scheme in which several computing methods are tried in sequence: static and semi-static filters, interval arithmetic, and multi-

precision algorithms for the edge cases. In the same spirit as Shewchuk's work, the possibility of interrupting early the sequence of computations, and only having to compute the exact solution in the most rare cases is crucial in order to attach a good average performance.

The static and semi-static filters are automatically generated from the user's source code, and it is expected for them to be successful almost always (when the predicate output is far from zero). Otherwise libraries for interval or multi-precision arithmetic are used; the latter is likely to be the slowest implementation available, but it provides always the exact result. Note that this is done in a case-by-case basis, so the extra cost of using sparingly an exact multi-precision algorithm is amortized with the rest of the computations.

4.2.2. Implementation of surface trimming for meshes

In order to accommodate the tasks of identifying intersecting and the trimming of the facets in the code, a new C++ library, LIM Collision Detection (LIMCODE) was written. This library is similar to some others already available as free software, but features a number of additional characteristics:

1. It does not require the meshes to be in a specific format, since the user can choose to pass the library either one reference to the list of vertices defining the shape, or a list of non-repeating vertices and a list of indices. The library does not store the data itself, it only uses it for building its internal structures.

2. The type of the data passed can be configured by the user, so single or double precision data can be passed without problems, or any other data type that features the common algebraic operators $(+, -, *, /)$.

3. Any multi-precision data type can be configured in order for the library to compute the exact value of the predicates only in the cases where it is near zero. Again, the only requirement on that data type is that defines the arithmetic operations specified in the previous item.

The first stage is commonly referred to as "far detection"; its aim is to discover how many pairs of objects are in potential contact. It usually consists of a simple test that quickly discards objects that are separated by a big distance, therefore avoiding to further spend computation cycles. Typically a simple volume, such as a sphere or a rectangular box is created and resized to enclose the whole object. Those fast tests can result in false positives, since the enclosing volume is bigger than the object itself, but never in false negatives. In a typical situation, the

number of false positives should be very low, and nevertheless processing them should be also very fast.

For virtual assembly purposes, the low amount of moving parts makes it possible to almost ignore this *far detection* stage, since the small number of checks do not constitute any significant computation load for the system.

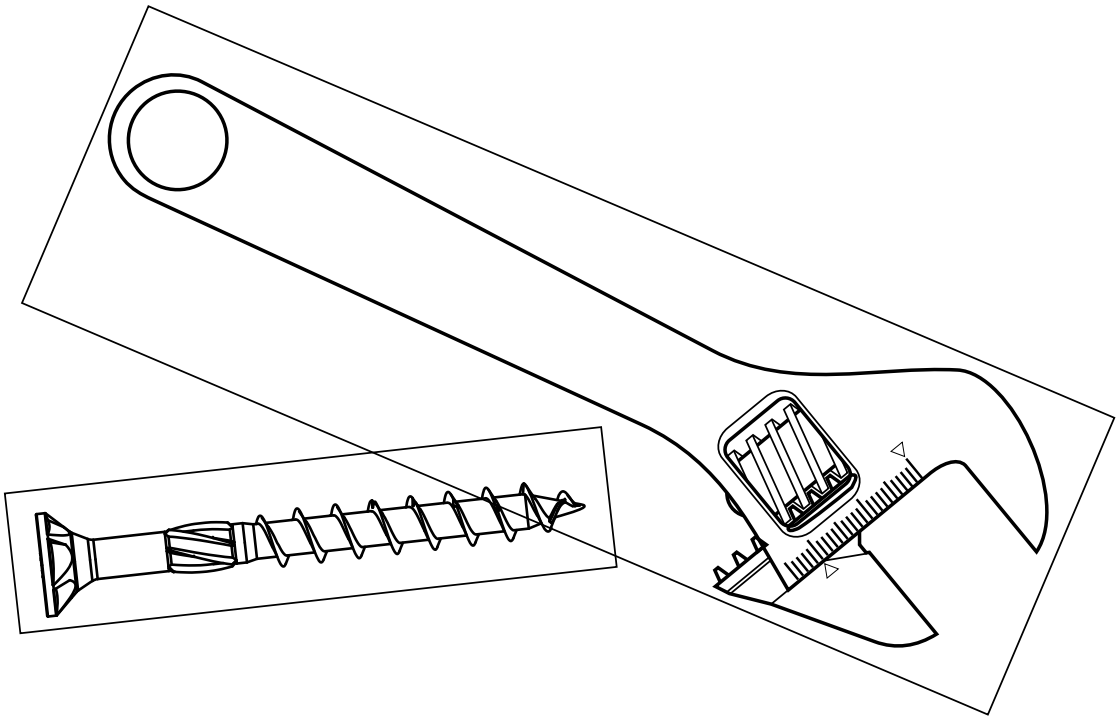

Figure 4.2: False positive when colliding bounding volumes (Wikipedia.org, swg-produktion.de)

4.2.2.1. AABB trees

After having the list of pairs that are potentially in collision, the system proceeds to what is commonly called "near detection" phase. In this stage, the collision is computed at a precise detail level; in the case of triangle meshes, the process finds all the pairs of intersecting facets. That information is needed to obtain the resulting intersection between the objects and its volumetric properties.

Contrary to what it was described for the *far detection* stage, the number of facets in any of the objects is usually large; checking all the possible combination of facets in the search for intersections would lead to a huge number of tests whose result is false. Even for offline boolean algorithms, this process is optimized through the classification of each facet into a spatial hierarchy that makes it possible to

quickly traverse among all the ones in its vicinity. A structure of that kind, used for interactive applications, is the Axis-Aligned Bounding Box (AABB) tree.

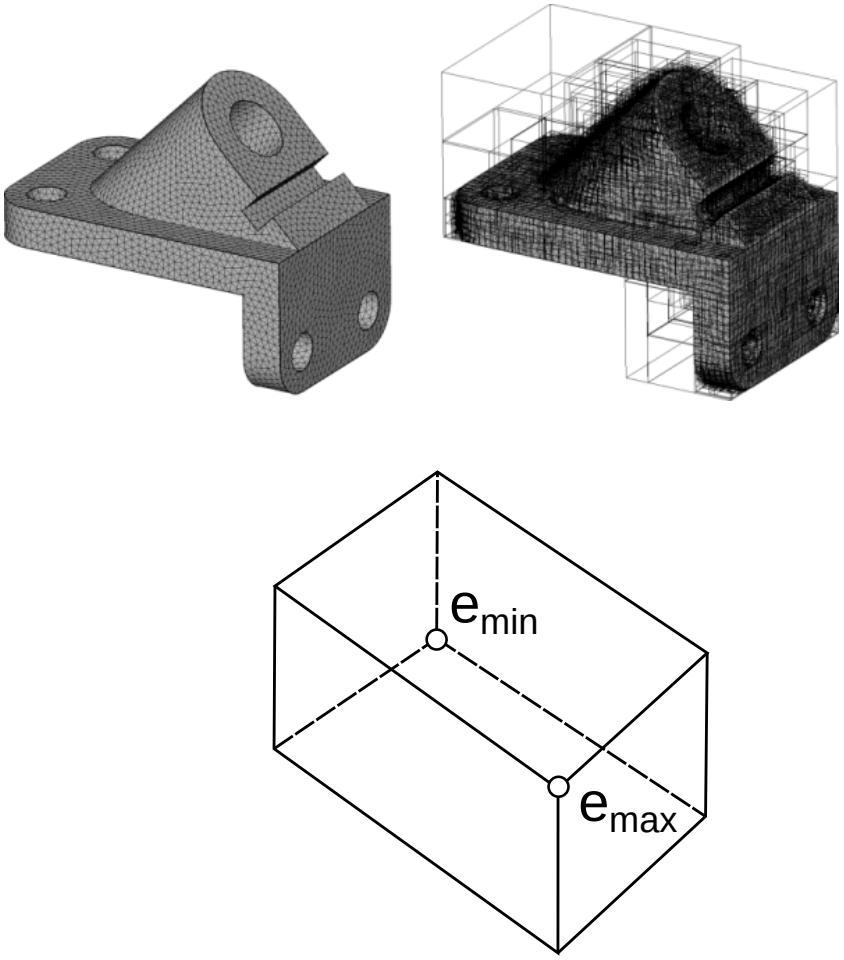

Figure 4.3: Representation of an AABB tree (cgal.org)

An AABB tree is a hierarchical structure composed of nodes, each of them enclosing a AABB volume; each of these volumes are rectangular boxes, aligned to the local axes X, Y, Z. Its only parameters are its extremal points $\mathbf{e}_{\min} = [x_{\min}, y_{\min}, z_{\min}]^T$ and $\mathbf{e}_{\max} = [x_{\max}, y_{\max}, z_{\max}]^T$. In order for the tree traversal to be as efficient as possible, each node should lead to their smaller-sized child nodes; at each level, the probability of intersecting all the children diminishes, so the number of discards increases, accelerating the process. There is not an unique algorithm for creating an AABB tree, since the criteria for classifying each group of facets at each level is not unique itself. However, any function splitting a group of facets into two equal-sized children groups would typically lead to good results, since it would diminish the complexity of a search from $O(n)$ to $O(\log(n))$.

The implementation of our C++ collision detection library, LIMCODE, follows a spatial policy in which the initial group is divided into two groups, depending on if a given facet lies on either side of a splitting plane. The splitting plane is computed to pass through the Center of Mass (COM) of the facet group, and its orientation to be perpendicular to the axis whose dimension is the largest. For each child group of facets, the process is repeated until each group holds exactly one primitive (triangle).

When performing the *near detection* stage between two solids, only their AABB trees are used. Considering that the affine transformation $\mathbf{T}_i \in \mathbb{R}^{4\times4}$ holds the translation and the rotation of each object i, the inputs of the procedure are $\mathbf{T}_A$ and $\mathbf{T}_B$. A rotated AABB is not aligned with the global coordinate system anymore, and it is called an Oriented Bounding Box (OBB). Even if checks on OBB entities are not very expensive in computational terms, they are still slower and less precise than checks on AABB boxes due to the higher number of floating-point operations involved. A typical optimization consists in expressing the problem in terms of the local coordinate system of one of the objects, therefore having to check an AABB tree against the OBB boxes of the other object. In that case, the relative transformation matrices $\mathbf{T}_A$ and $\mathbf{T}_B$ in the reference frame of the first object (A):

$$\mathbf{T}'_A = \mathbb{I}^{4\times4} \tag{4.3}$$

$$\mathbf{T}'_B = \mathbf{T}_A^{-1}\mathbf{T}_B \tag{4.4}$$

The *near detection* stage compares the nodes of each tree, checking if they are in collision. Since the process is carried in the frame of reference of one of the objects, the root AABB node of its tree is tested against the other object's AABB root node. The nodes of the second object have to be transformed into the frame of the first by applying $\mathbf{T}'_B$ to each AABB node, and the AABB-OBB test is performed. If the test is successful, the children of each node of each object have to be tested analogously. If a test fails, the node and its children are discarded, therefore pruning large parts of each tree.

The tree traversal ends at the primitive level, when two *leaf* nodes holding only one triangle each are tested.

The AABB tree collision test lessens the need for a complex *far detection* algorithm, since if the number of moving objects is moderate, non-colliding objects are quickly discarded at the first check of the root nodes of each object.

4.2.2.2. Testing of potentially colliding triangles

For these primitive tests, the method described by Guigue [30] is used. This algorithm decomposes the problem into a combinatorial stage and a predicate

evaluation, therefore containing the numerical error into a single place which can be modified to use different precision data types if necessary. The combinatorial stage detects the relative position of the vertices of one triangle with respect to the other, and then passes the ordered sequence to the orientation predicates. It is required to evaluate at most two predicates to know the results of the test. If the test is positive, the pair of indices of each triangle is stored for further processing.

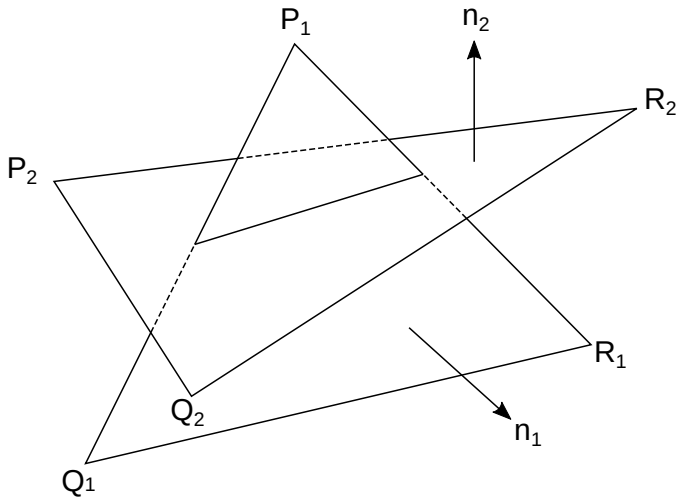

Figure 4.4: Vertex ordering in Guigue's method.

4.2.3. Intersection volume computation for meshes

After a collision between a pair of objects is detected, the system can proceed to identify the shape of the intersection enclosed by them (Figure 4.5). This procedure is composed of two different stages, depending on the type of primitive type:

1. Primitives detected in the collision detection: the triangles that were detected to intersect any other triangle of the second object (Figure 4.6). These primitives have to be clipped against the other object.

2. Neighboring primitives to the first type: triangles that touch those to be clipped and are inside any of the two objects. They have to be collected in order to close the intersection shape.

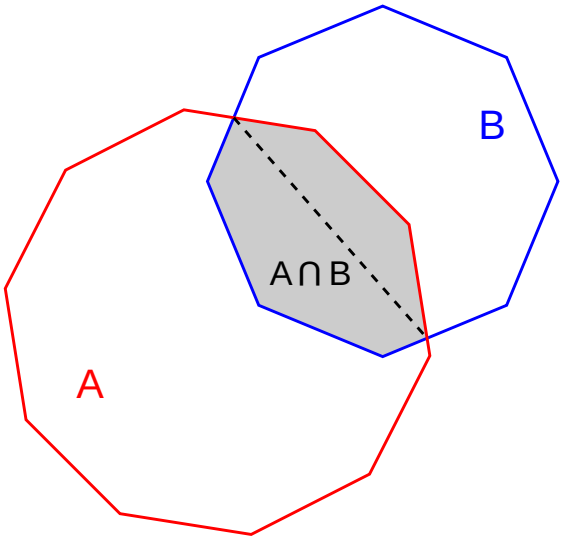

Figure 4.5: Intersection between a pair of objects

4.2.3.1. Binary Space Partitioning Trees

For both tasks, it is frequently needed to know if some entity lies in the inside or the outside of an object's boundary. As described in Chapter 4, an object is built by free-form surfaces whose only requirement is to define a manifold domain. Thus, simple local tests on the surface of an object can lead to erroneous results (Figure 4.7). This problem is caused by the fact that a shape is not required to be convex, so in a volume domain $\mathcal{V}$ and two points i and j such as $i, j \in \mathcal{V}$, any given point

$$k = i + \alpha(j - i) \quad \text{with } 0 \leq \alpha \leq 1$$

is not guaranteed to belong to $\mathcal{V}$ as well.

Thibault [57] develops a structure that decomposes a closed domain into a set of convex cells, called BSP. In the same spirit of the AABB trees, a BSP can be computed in the preprocessing step of a simulation, and it remains valid even the object is transformed by any affine operation, since it is referred to the local coordinate system. The procedure of building a BSP consists on successively splitting the mesh by a plane until each sub-mesh is a convex cell. An object has not a single BSP representation, since it depends on the plane selection used. This selection can alter the performance of the queries of the BSP, since they can result in better balanced or shorter trees. An usual strategy is to use the facets of the object itself as the partitioning planes for creating the BSP, thus reducing the number of cells in the tree: if a partitioning plane passes through any other facet, it has to be split and classified into both sides.

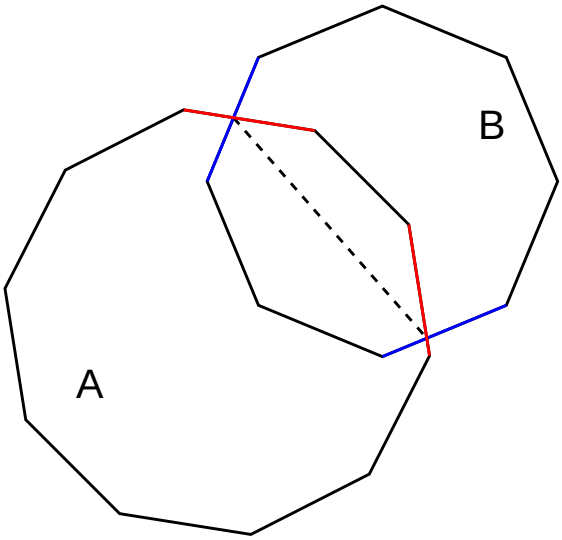

Figure 4.6: Intersecting faces detection stage

The procedure for determining if an entity lies in the inside of a given object is analogous to other binary-tree traversals: the entity is checked against the root node, and then classified as being in the positive ($\oplus$) or the negative ($\ominus$) sub-space that the node defines. The traversal continues by following the corresponding sub-space node at each generation until a *leaf* node is found: the classification of the entity against the whole object is the one obtained at the *leaf* node.

In the case of the BSP the classification has always to reach a *leaf* node, since they represent the convex cells in which the volume domain is de-composed, but since the path of the traversal has no bifurcations, it delivers very good performance, being $O(\log(n))$ as well.

4.2.3.2. Polygon Clipping

In the process of creation of a BSP, or in the mesh intersection stage, some polygons are required to be cut against a given plane (Figures 4.9 and 4.11). Splitting a polygon in two different pieces is a brittle numerical task if performed naively; as presented in [20], numerical errors can lead to inconsistent output (the result set of split polygons does not cover the same domain as the original one, open polygons, ...)

Again, geometrical predicates are used in order to contain and identify the potential source of numerical errors, and to be able to exploit different computational strategies depending on the difficulty of the problem.

The algorithm proposed by Bernstein [5] is able to compute the split polygons corresponding to one of the sides of the splitting plane in only one traversal of

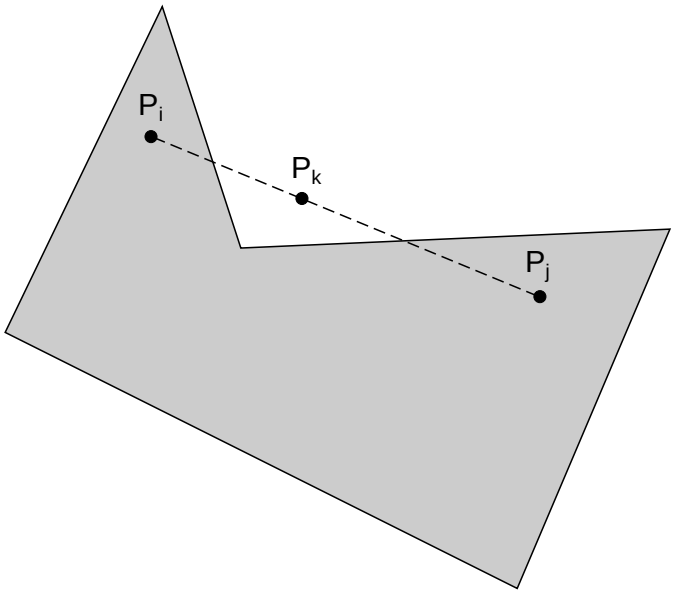

Figure 4.7: Definition of a non-convex domain

the original polygon. To accomplish the task, it follows each vertex in order and classifies them against the splitting plane, using the predicates.

Depending on the classification of a vertex and the classification of the previous 2 vertices, it can be known if a vertex belongs to the polygon in the positive side of the plane, or the negative, or both. Since it is required to store at most the information from the previous pair of vertices in the sequence, it is straightforward to implement the algorithm as a Finite State Machine (FSM), an algorithm that has a predefined set of states, and has rules to enforce that the nodes can only be followed in some allowed sequences.

For this matter, the state of the clipping algorithm is defined by the classification results of the last three vertices against the splitting plane. The possible states of the FSM are shown in Table 4.1: $\ominus$ determines a point is below the plane, $\oplus$ that is above it, "0" strictly on the plane, and "*" means that the classification of that point is not relevant.

The output corresponding to each of the stages is marked as "H" (the splitting hyperplane) or "B" (boundary hyper-plane), depending on which new boundary the algorithm is creating: if it is using the splitting plane or if it continues to use the current boundary of the polygon. In the case that a joint mark is output (i.e. "HB"), it means that the code is emitting both planes, and in that precise order. Outputs marked with an empty set ($\emptyset$) mean that the algorithm should proceed to the next vertex without adding any new boundary.

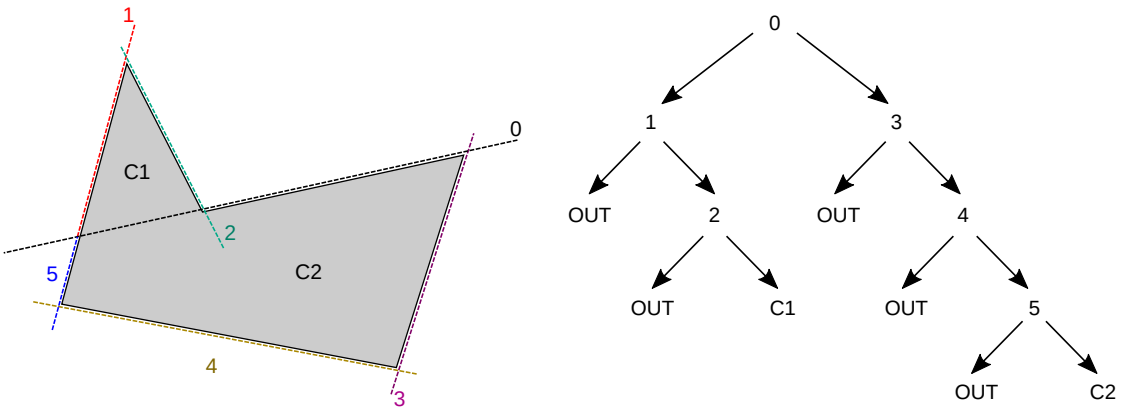

Figure 4.8: BSP tree. *Leaf* nodes point to convex cells.

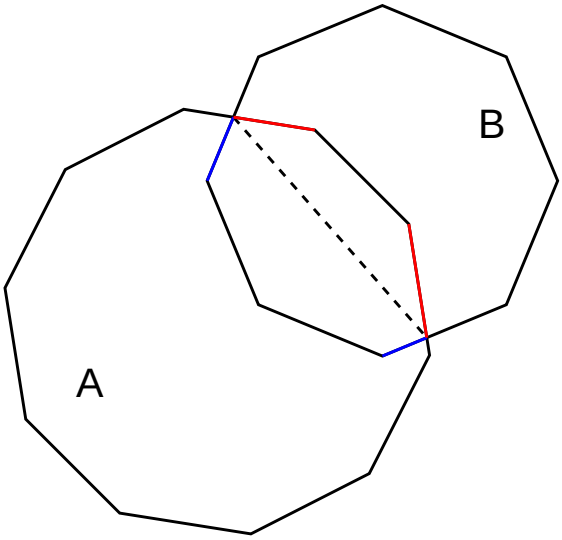

Figure 4.9: Clipping faces stage

4.2.3.3. Half-edge structure

Once the intersecting facets from both objects are clipped against the other object, the intersection shape has to be completed by adding the rest of the facets that lie in the inside of the objects (Figure 4.12). This task can be precomputed by storing beforehand the connectivity information of each facet in the preprocessing stage. LIMCODE uses a *half-edge* structure [37] which encodes the local information of each facet and makes it possible to traverse the mesh following some criteria, as cycling through the vertices of a facet, or jumping to an adjacent facet.

The structure is composed of a collection of edge nodes $N_e(V, F, N'_e)$, each one holding a reference to the vertex V to which the edge is pointing, to the face F

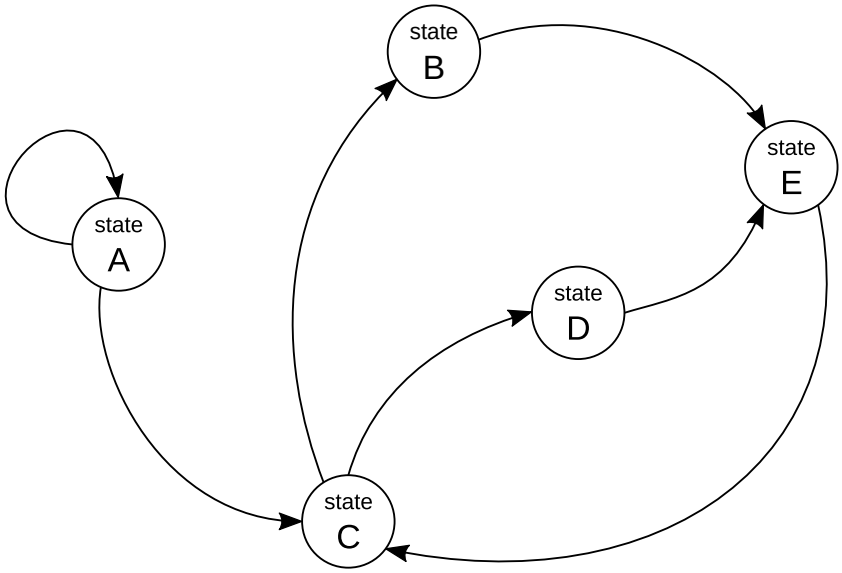

Figure 4.10: Depiction of a FSM: "A" is the starting node and valid transitions are marked with arrows.

Input	Output	Input	Output	Input	Output
$*\oplus\oplus$	B	$\oplus 0\oplus$	B	$*\ominus\oplus$	HB
		$00\oplus$	HB		
		$\ominus 0\oplus$	HB		
$*\oplus 0$	B	$*00$	$\emptyset$	$*\ominus 0$	$\emptyset$
$*\oplus\ominus$	B	$*0\ominus$	$\emptyset$	$*\ominus\ominus$	$\emptyset$

Table 4.1: Possible states and results of the FSM

which it belongs, and to the coincident, opposite-direction edge that belongs to the neighboring face N'_e. As the mesh is required to be manifold, only one node N'_e is opposed to the node N_e, and it is also guaranteed that each edge node has a complementary node; otherwise the surface would be open.

Precisely that complementary edge node N'_e allows to traverse the surface along the adjacent facets of a given one. In order to close one of the caps of the intersection surface starting from the intersection primitives, an interior face F_{wi}, neighbor to the intersection face F_i has to be found, such as

$$F_{wi} \in N'_{e1}, N'_{e2}, N'_{e3} \text{ from } F_i/\text{BSP}_B(F_{wi}) < 0$$

the search stops when the interior face of object "A" (resp. "B") is classified as "interior" by using the BSP structure of the opposite object "B" (resp. "A").

The rest of the algorithm is very simple: a list containing the intersection faces

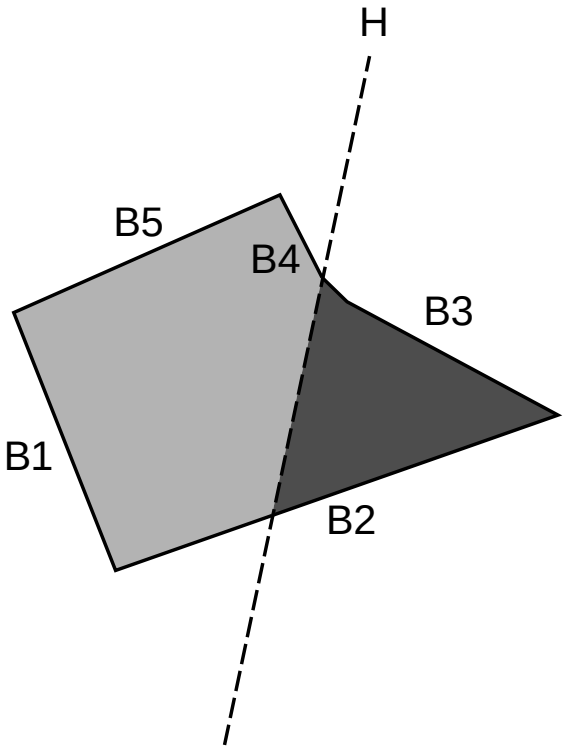

Figure 4.11: Representation of the boundary planes B_i (edges) of the polygon and the splitting plane H

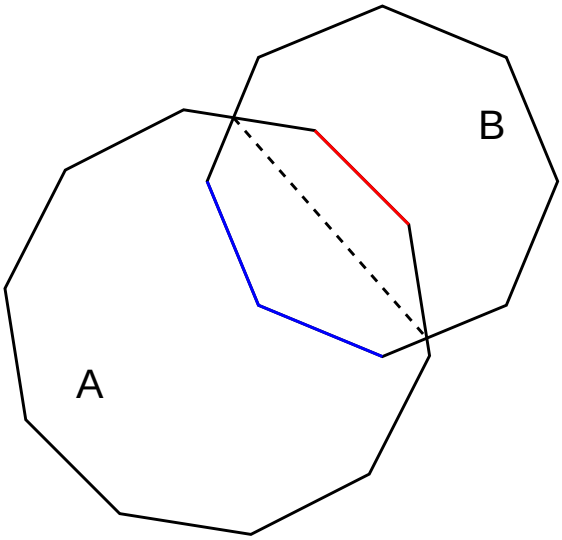

Figure 4.12: Internal faces detection stage

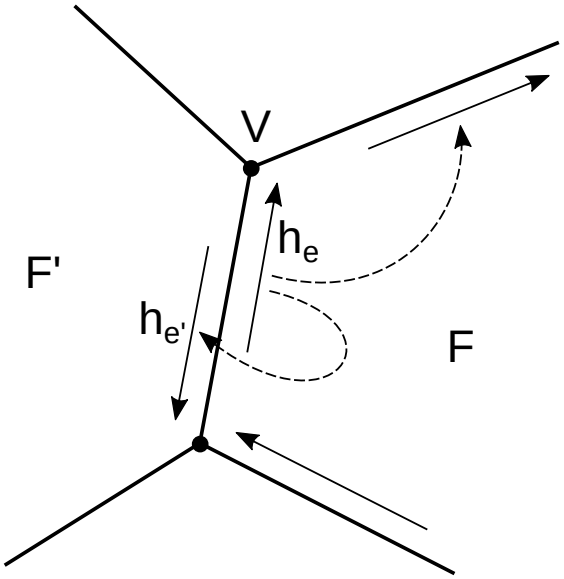

Figure 4.13: The *half-edge* structure.

F_i and the interior face F_{wi} is created, L_{if}. Then, the algorithm traverses all the neighbors of F_{wi} not present on L_{if} and adds them to it; the algorithm continues the traversal until it cannot advance anymore: all the forward faces are already present in L_{if}. The interior surface cap is guaranteed to be continuous and gapless, because the imposed manifold property of the objects: the algorithm cannot traverse the initial ring of intersection faces F_i, and there has to exist a valid path from them to any of the interior faces F_{wi}.

4.2.3.4. Computation of volume properties

Once the intersection boundaries of both objects are defined, they form a closed, manifold object whose solid properties can be computed (Figure 4.14). The contact model requires to know three mass-moments: the volume of the object, its COM and the inertia tensor with respect to it.

Some methods have been developed to accomplish that task; one of the most cited ones is Mirtich's [42], which can deal with manifold meshes of polygons, each one with an arbitrary number of sides. The algorithm takes advantage of the successive application of the Divergence Theorem to a closed surface to transform the volume integral into a bi-dimensional one per facet. In the same way, Green's Theorem is used later to finally reduce each of the area integrals into line integrals.

As an example of how the method works, it is shown the procedure of computing one of the products of inertia of a solid, $P_{xy} = \int_{\mathcal{V}} xy\mathrm{d}V$ over its volume domain, $\mathcal{V}$.

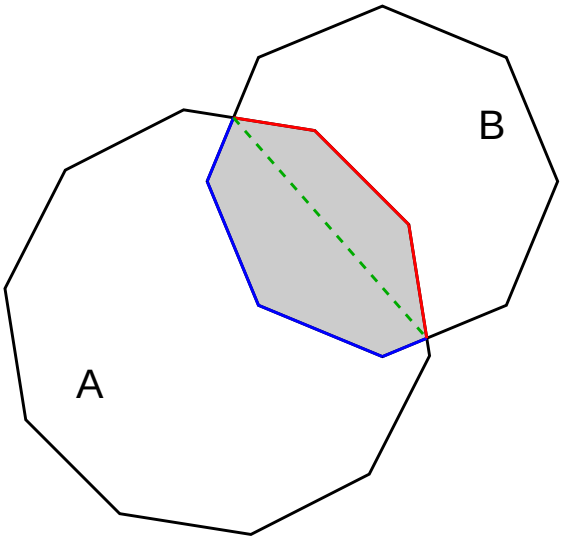

Figure 4.14: Complete intersection volume

The Divergence Theorem states that

$$\int_{\mathcal{V}} \nabla \cdot \mathbf{F} \, \mathrm{d}V = \int_{\partial\mathcal{V}} \mathbf{F} \cdot \hat{\mathbf{n}} \, \mathrm{d}A$$

The method by Mirtich takes advantage of the fact that if a function $\mathbf{F}$ is found such as $\nabla \cdot \mathbf{F}$ is the desired quantity to obtain, the volume integral over the whole solid can be decomposed into several, simpler area integrals, one per facet (the sub-domains $\partial\mathcal{V}$). For example, it can be checked that a function as $\mathbf{F}_{xy} = 1/2x^2y\mathbf{i} + 0\mathbf{j} + 0\mathbf{k}$ complies with the requirements to compute the integrand of the product of inertia P_{xy}, as $\nabla \cdot \mathbf{F}_{xy} = xy$. Then, the volume integral can be transformed into a collection of area integrals

$$P_{xy} = \int_{\mathcal{V}} xy \mathrm{d}V = \int_{\mathcal{V}} \nabla \cdot \mathbf{F}_{xy} \, \mathrm{d}V = \int_{\partial\mathcal{V}} \mathbf{F}_{xy} \cdot \hat{\mathbf{n}} \, \mathrm{d}A = \frac{1}{2} \sum_{\mathcal{F} \in \partial\mathcal{V}} \hat{\mathbf{n}} \int_{\mathcal{F}} \mathbf{F}_{xy} \, \mathrm{d}A \qquad (4.5)$$

for each facet $\mathcal{F}$. Note that, since the facets are flat, the normal $\hat{\mathbf{n}}$ of each surface is constant and does not need to be integrated. Analogously, Mirtich uses the Green's Theorem to express each of the area integrals in terms of line integrals,

$$\int_{\mathcal{F}} \nabla \cdot H \mathbf{d}A = \oint_{\partial\mathcal{F}} H \cdot \hat{\mathbf{m}} \mathbf{d}s \qquad (4.6)$$

where H is a function that, in the same sense as F, aids to transform the area integral into a collection of line integrals along each of the edges of the polygon that encloses that area and whose outside-pointing normal is $\hat{\mathbf{m}}$.

After collecting all the terms for each edge and polygon facet, the properties can be added and transmitted to the next part of the program. This process is simple, but it requires a significant amount of floating-point operations, therefore accumulating numerical errors in the process, which could be noticeable when the volume is small.

If the mesh to be processed is composed only by triangles, the method can be further simplified. Since any flat polygon can be decomposed into a set of triangles, only a small preprocessing overhead would be required. Eberly [19] reports that simplifications stemming from the only use of triangles "require significantly less computational time" than the original algorithm.

DiCarlo [13] suggests a much simpler implementation by using an affine extension to the Euler tensor. This method can be used also with affine operations, so the final results can be obtained without having to transform them afterwards. Since only multiplications and additions are required to implement the method, is well-suited to be used in Single Instruction Multiple Data (SIMD) computing devices such as any Graphics Processing Unit (GPU).

The method extends the notion of the Euler tensor $\mathbf{E} \in \mathbb{R}^{3\times3}$ to an augmented $\mathbf{E}_+ \in \mathbb{R}^{4\times4}$:

$$\mathbf{E} = \int_B \mathbf{r} \otimes \mathbf{r} \mathrm{d}M \Rightarrow \mathbf{E}_+ = \int_B \mathbf{r}_+ \otimes \mathbf{r}_+ \mathrm{d}M \tag{4.7}$$

where $\mathbf{r}$ is a vector pointing to each $\mathrm{d}M$ of the body, and $\mathbf{r}_+$ is the equivalent variable in homogeneous coordinates.

Developing 4.7, and upon closer inspection of 4.8, it is clear that the upper-left corner of $\mathbf{E}_+$ holds the inertia tensor, the last three elements of the last row or column the COM, and the element at (4, 4) the mass, since the homogeneous coordinate $w = 1$ for points.

$$\mathbf{E}_+ = \int_B \begin{pmatrix} x^2 & xy & xz & xw \\ xy & y^2 & yz & yw \\ xz & yz & z^2 & zw \\ xw & yw & zw & w^2 \end{pmatrix} \mathrm{d}M \tag{4.8}$$

The method computes the contribution to the tensor of each triangular facet with the following expression

$$\mathbf{E}_+ = \det(\mathbf{G})\mathbf{G}\mathbf{E}_+^*\mathbf{G}^T \tag{4.9}$$

where $\mathbf{E}_{+}^{*}$ is a constant tensor of the form

$$\mathbf{E}_{+}^{*} = \frac{1}{120}\begin{pmatrix} 2 & 1 & 1 & 5 \\ 1 & 2 & 1 & 5 \\ 1 & 1 & 2 & 5 \\ 5 & 5 & 5 & 20 \end{pmatrix} \tag{4.10}$$

and $\mathbf{G} \in \mathbb{R}^{4\times4}$ is a matrix holding the three coordinates $\mathbf{v}_i$ of a triangle vertex:

$$\mathbf{G} = \begin{pmatrix} \mathbf{v}_0 & \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.11}$$

All the results are referred to the origin of coordinates $(0, 0, 0)$.

4.3. Inner Sphere Trees volume trimming

Mesh analysis is usually a costly approach for calculating the intersection volume, because of the need to compute every colliding facet and clipping the intersecting ones requires a lot of processing time. Therefore, a simpler alternative method was also used to compare its performance and accuracy against the mesh intersection method. The alternative selected, known as IST, is described in [58] and its based on the ProtoSphere sphere packing algorithm presented on [59].

Basically, the method consists in using a simplified geometry for the object representation. Instead of using the triangle mesh to calculate the intersection volume, a group of non-overlapping spheres of diverse size, known as sphere packing, is used to fill the object and therefore to approximate its shape. Usually, around 10 000 spheres are enough to get a rough approximation, depending on the object's size and shape. This simplification makes it easier to find the intersection volume as no triangle checking is involved in the operation. Once the sphere packing is created using the ProtoSphere tool, a hierarchy is established between the spheres in order to speed-up the collision checks analogously to the aforementioned AABB trees. This hierarchy is called Inner Sphere Tree, and it is a top-down tree where the leaves (spheres from the sphere packing) are contained inside node spheres, which are also grouped together into bigger sphere nodes until only one sphere wraps the object entirely. This makes it possible to have a time critical algorithm that can check the approximate overlapping volume running in a predefined time budget.

On top of these tools, an algorithm was implemented in order to calculate the rest of the needed properties, such as the center of mass, inertia tensor of the intersection volume and the normal force direction vector. Since a collision between two objects is represented as a two Inner Sphere Trees intersection, the interference volume is approximated as a number of colliding pairs of spheres. The

overall mass properties can be derived then from the sum of the properties of the individual sphere pairs.

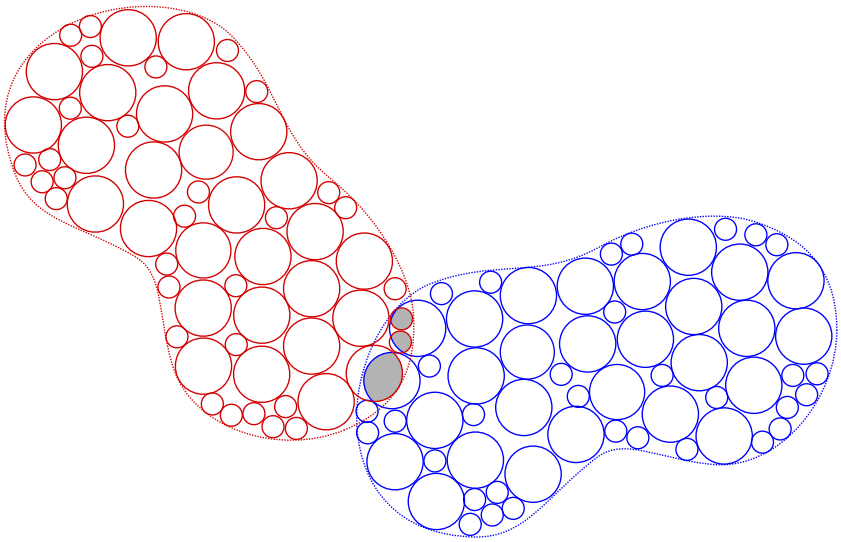

Figure 4.15: IST collision

4.3.1. Sphere intersection

Given two spheres A and B, of radius R_A and R_B, located at $\mathbf{p}_A$ and $\mathbf{p}_B$ respectively, the distance between the spheres can be calculated as $d = \sqrt{(\mathbf{p}_B - \mathbf{p}_A)(\mathbf{p}_B - \mathbf{p}_A)}$. A is assumed to be the biggest sphere, $R_A \geq R_B$.

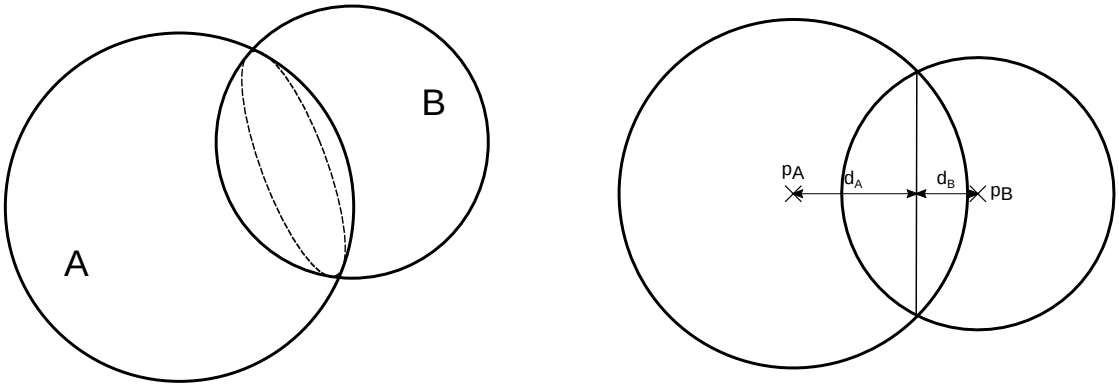

Figure 4.16: Collision of two spheres

The distance from the contact plane to the center of both spheres is

$$d_A = \frac{R_A^2 + d^2 - R_B^2}{2d} \tag{4.12}$$

$$d_B = \frac{R_B^2 + d^2 - R_A^2}{2d} \tag{4.13}$$

$$d = d_A + d_B \tag{4.14}$$

Depending on the relation between this distances and the radius of both spheres, we can have up to four different situations, as shown in Figure 4.17. In order to calculate the intersection volume V_i, its centroid $\mathbf{p}_i$ and its inertia tensor $\mathbf{I}_i$, the following cases must be tested sequentially until one that fits the current situation is found.

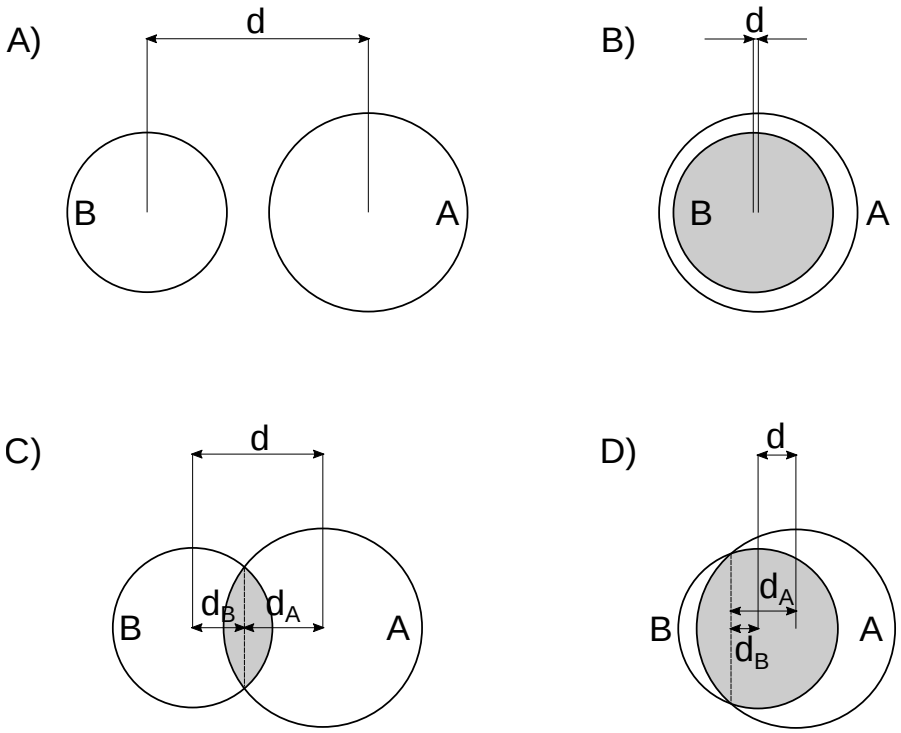

Figure 4.17: Sphere intersection types

A) $d \geq R_A + R_B$ There is no contact between the spheres.

$$V_i = 0 \tag{4.15}$$

B) $R_A \geq d + R_B$ Sphere B is completely inside sphere A

$$V_i = \frac{4\pi R_B^3}{3} \tag{4.16}$$

$$\mathbf{p}_i = \mathbf{p}_B \tag{4.17}$$

$$\mathbf{I}_i = \frac{2m_B R_B^2}{5}\mathbb{I}_3 \tag{4.18}$$

C) $d \geq d_A$ Less than half sphere B is inside sphere A. The intersection is the union of two spherical caps and their properties are calculated using the expressions in Section 4.3.1.1.

D) $d < d_A$ More than half sphere B is inside sphere A. This case is similar to Item C), but one of the caps is larger than half sphere, so despite the same expressions can be used, this has to be taken into account: for this cap properties to be calculated, first the properties of its complementary cap must be found and then subtracted from the properties of the whole sphere.

4.3.1.1. Spherical cap properties

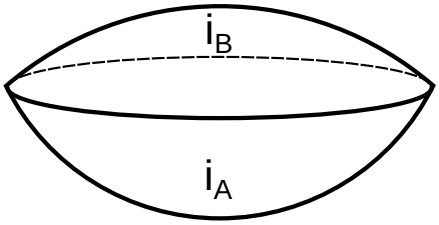

Figure 4.18: Spheres intersection

As previously stated, in the case that $d < R_A + R_B$, it can be inferred that there is penetration between both spheres. This interference i is usually a biconvex lenticular body that can be decomposed in two spherical caps (i_A and i_B) cut by the contact plane, the one which contains the contact circumference (Figure 4.18). If the collision is positive, the mass properties of the caps i_A and i_B can be calculated using the following formulas for a spherical cap where h is the height of the cap and R the radius of the sphere.

$$V_{cap} = \frac{\pi h^2}{3}(3R - h) \tag{4.19}$$

$$z_{ccap} = \frac{3(2R-h)^2}{4(3R-h)} \tag{4.20}$$

$$\mathbf{I}_{cap} = \begin{pmatrix} \mathrm{I}_{11cap} & 0 & 0 \\ 0 & \mathrm{I}_{22cap} & 0 \\ 0 & 0 & \mathrm{I}_{33cap} \end{pmatrix} \tag{4.21}$$

$$\mathrm{I}_{11cap} = \mathrm{I}_{22cap} = \frac{\pi h^2}{60}(60R^3 - 9h^3 - 80hR^2 + 45h^2R) \tag{4.22}$$

$$\mathrm{I}_{33cap} = \frac{\pi h^3}{30}(20R^2 + 3h^2 - 15hR) \tag{4.23}$$

Where V_{cap} is the volume of the cap, z_{ccap} is the height of its centroid along its symmetry axis and $\mathbf{I}_{cap}$ is the inertia tensor, all of them expressed in the center of the sphere.

Once the mass properties have been calculated for every sphere pair, all of them can be added-up to get the total values for the whole set properties.

4.3.2. Numerical error optimization

While the expressions above are theoretically correct, there is still a problem with their implementation. As every calculation made on a computer, this equations suffer from some floating-point errors that may render them inaccurate. Truncation error or loss of significance effects like catastrophic cancellation are some issues that affect the results. In some other cases this errors may not be significant, but when trying to discern between contact and no-contact situations (or any other boolean output), a false positive due to numerical error leads to division-by-zero calculations and other undesirable issues.

Let's say for example that while evaluating Equation (4.12) or Equation (4.13) a slightly inaccurate result is obtained that leads to a positive output for the collision test described before when the two spheres are not in contact or just tangent. This would yield a negative value for h and would be propagated to Equation (4.19), (4.20) and Equation (4.21) producing also negative values for V_{cap}, z_{ccap} and $\mathbf{I}_{cap}$, which has absolutely no physical meaning and would make the force model fail and therefore invalidate the simulation.

In collision-detection-related equations, minimizing the floating point error is critical because a lot of terms can produce wrong results leading to an incorrect contact detection, being catastrophic for the simulation. The equations above, in their current form, are numerical error prone, so they must be reworked in order to avoid these issues. In [47], a heuristic algorithm that reformulates equations to

minimize their floating-point error is presented. This tool, called Herbie, was used to generate new, improved equations.

Herbie works sampling random input points for the given equations over all the floating-point range and identifying known errors to certain mathematical operators. It then rewrites the expression using a heuristic search algorithm to find programs that compute the same equation more accurately. Series expansions are used to avoid under/overflow and all the different paths that branch-out from the original are combined into an optimized solution. Sometimes this solution is a set of conditional expressions, and the one that must be evaluated will depend on the numeric value of the input.

The improvement in Herbie's reformulation process can be measured with the average bit error. This accuracy error is measured in Units in the Last Place or Units of Least Precision (ULP) and represents the spacing between floating-point numbers. It can also be understood as the value the least significant digit represents if it is 1.

The new equations generated by Herbie are

$$d_A = 0.5\frac{R_A}{\frac{d}{R_A}} + 0.5d - 0.5\frac{R_B}{\frac{d}{R_B}} \tag{4.24}$$

$$d_B = 0.5\frac{R_B}{\frac{d}{R_B}} + 0.5d - 0.5\frac{R_A}{\frac{d}{R_A}} \tag{4.25}$$

$$V_{cap} = (h(\pi h))(R - \frac{h}{3}) \tag{4.26}$$

$$z_{ccap} = \frac{(h0.75)h - (3R)(h - R)}{3R - h} \tag{4.27}$$

The distance average bit error using this new expressions goes from 26.7 with the old equations all the way down to 0.2 with the new ones. This can be appreciated in Figure 4.19, Figure 4.20 and Figure 4.21. These charts represent the bit error for the old and also the new expressions in all the floating-point range versus the different input values. For the sake of brevity, only the charts correspondent to d_A (4.12) reformulation are included, as there is one figure for every input R_A, R_B and d.

In the case of the intersection volume, the average bit error is 5.8 with the old formula and 0.4 with the new one. For the centroid, the bit error improves from 30 to 0.3.

It can be seen that this poses a great error improvement and certainly it was a key determinant factor in solving the false positive collision situations.

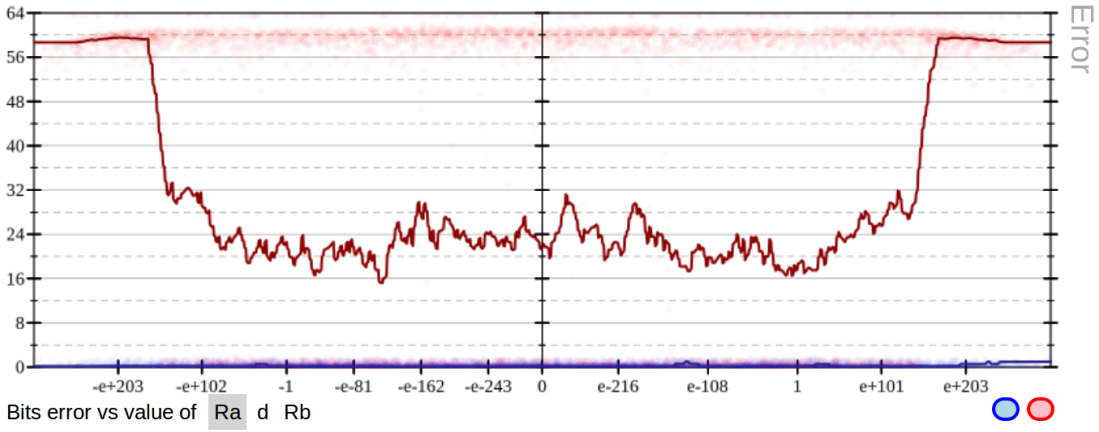

Figure 4.19: Herbie output chart: d_A expression bit error vs value of R_A, old (red) and new (blue)

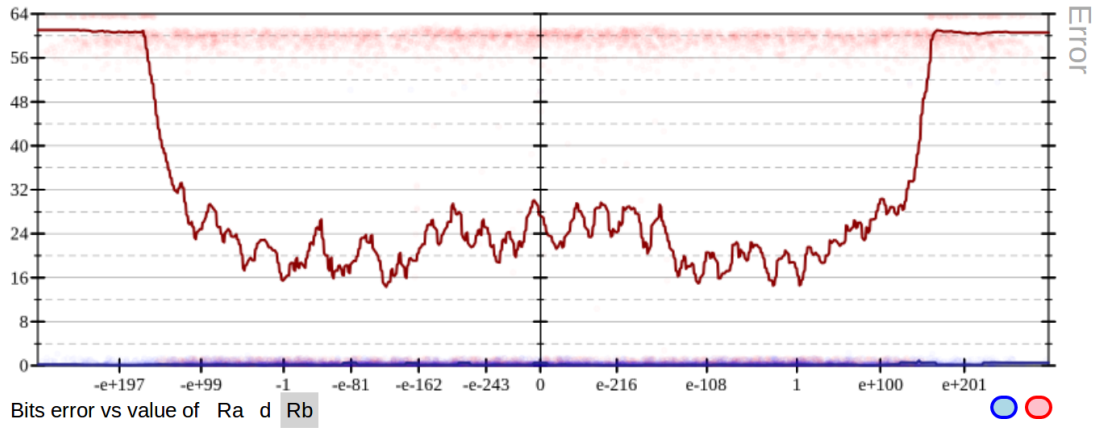

Figure 4.20: Herbie output chart: d_A expression bit error vs value of R_B, old (red) and new (blue)

4.3.3. Computation of the derivatives of the contact properties

Some of the collisions that happen during simulations are impacts: low duration, high frequency forces that vastly affect the integrator stability and therefore the simulation robustness. Introducing these forces can make the integration process require a high number of iterations in the search of a valid solution, and put at risk the real-time requirements. In order to improve the convergence of this process, the collision forces contribution to the tangent matrix in Equation (2.20) must

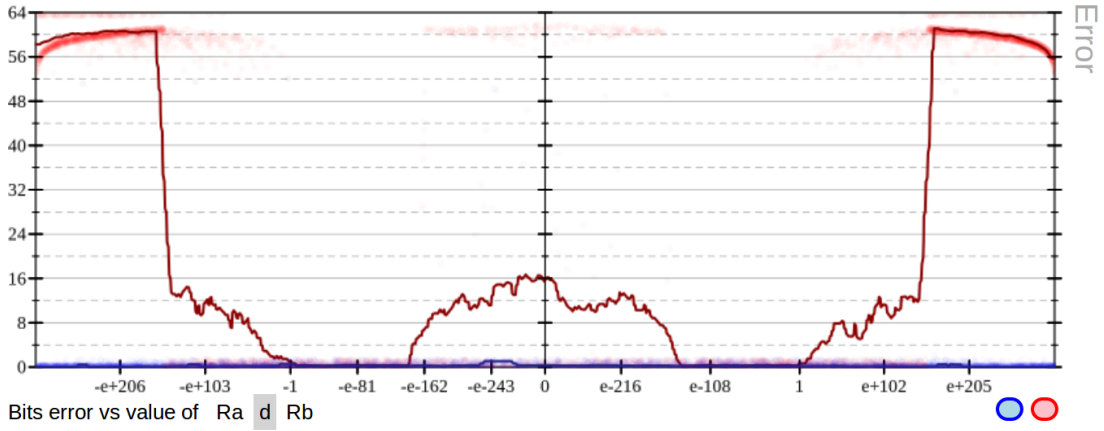

Figure 4.21: Herbie output chart: d_A expression bit error vs value of d, old (red) and new (blue)

be also calculated. As stated there, the contribution for a force is $\mathbf{K} = -\partial\mathbf{Q}/\partial\mathbf{q}$ and $\mathbf{C} = -\partial\mathbf{Q}/\partial\dot{\mathbf{q}}$.

This contribution can be analytically deducted for simple force models, for which derivatives with respect to $\mathbf{q}$ and $\dot{\mathbf{q}}$ can be found. For more complex force expressions, such as medium-sized mathematical expressions or piecewise-defined functions, their derivatives could not be easily found or they could not exist in the general case.

For the geometric model proposed in this section,

$$\mathbf{Q} = \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}, V(\mathbf{q}), \mathbf{I_A}(\mathbf{q}), k_n, h_n, \ldots) \tag{4.28}$$

In addition to the fact that the complete expression for 4.28 is not short, the functions in which they are based are defined differently depending on the relative position of each pair of spheres, and therefore the differentiation algorithm has to take that into account.

The simplest approach is to approximate the value of the derivatives numerically by the at first sight straightforward "finite difference" method: evaluate the function f in the vicinity of the point of interest, $f(\mathbf{x}_o)$ and in another close point that is at a small distance ε, $f(\mathbf{x}_o + \varepsilon)$. The quotient $(f(\mathbf{x}_o) - f(\mathbf{x}_o + \varepsilon))/\varepsilon$ is the definition of the derivative as $\varepsilon \to 0$. However, the subtraction of very close numbers is specially problematic in floating point arithmetic, and therefore the implementation has always to balance the effects of choosing a big ε parameter, which would lead to an inaccurate computation, or a small ε, which would lead to inadmissible numerical errors.

For overcoming these problems, a common alternative approach derivatives is

to use the "complex step" method [39, 54]. It is analogous to the previous one, but in this case a complex perturbation $i\varepsilon$ is chosen, instead of a real number. The expression to evaluate the derivatives is shown in Equation (4.29). The resulting effect is that the computed derivative is much less dependent on the size of the perturbation, and the numerical error is noticeably smaller.

$$f'(\mathbf{x}_0) \approx \frac{\mathrm{Im}(f(\mathbf{x}_0 + i\varepsilon))}{\varepsilon} \tag{4.29}$$

There are still some disadvantages when using the "complex step" method in a generic software implementation:

1. existing code performing the force evaluation has to be rewritten in order to use complex arithmetic;

2. the whole evaluation process has to be carried by this method: it is not possible to use the derivatives that can be provided by "black box" elements in terms of which the force function evaluation could be based.

A third method for evaluating derivatives is the family of "automatic differentiation" procedures. Those methods consist in the evaluation of the derivatives of each of the operations that define the expression, and accumulating the results for incorporating them to the subsequent calculations by means of the "chain rule". The evaluation of the derivatives is performed at the same time as the evaluation of the function itself. The derivative of each elementary term is either provided by the user, or considered null by default. Therefore, "black box" models, for which the algorithm is not available, can be used seamlessly as long as their derivatives are known, thus addressing Item 2. Some automatic differentiation systems require the function to be defined in terms of a special number type, but some others rely in code transformation, therefore addressing Item 1.

This system is based on the C++ Eigen library [29]. Eigen is a templating library implementing linear algebra algorithms. The "templating" term refers to the fact that those algorithms are written in an independent way from the data type, and thus can handle many numeric formats, such as single, double precision or complex ones. This is the same philosophy shown in the library described in Section 4.2.2. The library also includes an automatic differentiation capabilities module (Auto Diff), which is activated by using the special `Eigen::AutoDiffScalar` type in the function evaluation.

Listing 4.1 illustrates the usage of the module in the computation of the terms needed for calculating a force and the derivatives of each one of them. Each component is augmented with an array holding its derivatives with respect to (q_x, q_y, q_z). By default they are considered null, meaning the parameter is a constant. Lines 18 to 20 show how to declare a variable directly dependent on the set of three

coordinates, `cA`. This scheme would be used as well to link to any procedure that could supply the evaluation of a function as well as the derivatives, but not the actual implementation.

Following the code, it can be seen that the calculations are performed in a normal way, and that they spawn newer automatic differentiation variables that accumulate the derivatives as well the function evaluation.

As an example, the contact normal is stored in the variable named `direction`, and the lines 56 to 58 show that the actual computed value can be retrieved with the method `value()`, while the three derivatives of each component of the vector can be found with the method `derivatives()` (see lines 60 to 62).

Listing 4.1: Usage of the automatic differentiation module

```
1
2  // Type declaration:
3  // diffcomp: scalar value and its three derivatives (q_x, q_y, q_z)
4  typedef Eigen::AutoDiffScalar<Eigen::Vector3f> diffcomp;
5  // V3df: vector of three "diffcomp" elements
6  typedef Eigen::Matrix<diffcomp, 3, 1> V3df;
7
8  // cA, cB: regular vectors holding the coordinates of the spheres
9  Eigen::Vector3f cA(pA.getX(), pA.getY(), pA.getZ());
10 Eigen::Vector3f cB(pB.getX(), pB.getY(), pB.getZ());
11 // centerA, centerB: augmented vectors with automatic derivatives.
12 V3df centerA, centerB;
13
14 // cA and cB are transformed into the global frame of reference.
15 // centerA and centerB are loaded with them; sphere A is supposed
16 // to be moving, while sphere B is considered fixed.
17 cA = TA * cA;
18 centerA << diffcomp(cA[0], Eigen::Vector3f(1, 0, 0)),
19            diffcomp(cA[1], Eigen::Vector3f(0, 1, 0)),
20            diffcomp(cA[2], Eigen::Vector3f(0, 0, 1));
21 cB = TB * cB;
22 centerB << cB[0], cB[1], cB[2];
23
24 // Distance between centers
25 diffcomp dist = sqrt((centerB - centerA).dot(centerB - centerA));
26
27 // Check that there is actually contact
28 if(dist >= (RA + RB)) return;
29
30 diffcomp vol;
31 Eigen::Vector3f com;
32 Eigen::Matrix3f Ig = Eigen::Matrix3f::Zero();
33
34 // Unit vector from pA to pB
35 V3df direction;
```

```
36 // direction.normalize();
37 direction = (centerB - centerA) / dist;
38
39 // Check if B is completely inside A
40 if( (dist + RB) <= RA )
41 {
42   // Volume and Center Of Mass computation
43   vol = 4 * M_PI * pow(RB,3) / 3;
44   com << centerB[0].value(), centerB[1].value(), centerB[2].value();
45
46   // Computation of the inertia tensor of the volume
47   float i = 2 * vol.value() * RB * RB / 5;
48   Ig.diagonal() << i, i, i;
49
50   // Regular vector holding the direction of the force
51   Eigen::Vector3f normal;
52   // 3x3 matrix holding the derivatives of (n_x, n_y, n_z) w.r.t. (q_x, q_y, q_z)
53   Eigen::Matrix3f dn_dr;
54
55   // value() holds the actual value of a given component
56   normal << direction[0].value(),
57             direction[1].value(),
58             direction[2].value();
59   // derivatives() holds the three derivatives of a given component
60   dn_dr << direction[0].derivatives(),
61            direction[1].derivatives(),
62            direction[2].derivatives();
63
64   // Use the force in the multibody system
65   force_callback(vol.value(),
66                  com.data(),
67                  normal.data(),
68                  Ig.data(),
69                  vol.derivatives().data(),
70                  dn_dr.data(),
71                  idA,
72                  idB,
73                  nCollisions);
74
75   return;
76 }
```

Chapter 5

Results

To assess the correctness and the performance of the presented contact methods, three different tests were designed, each one focused on studying the model behavior during one of the possible relative motions between two colliding bodies: sliding, rolling and spinning.

5.1. Test 1: block sliding on plane

5.1.1. Description

The first of the performed tests was designed to verify the accuracy of the normal and tangential force models. It consisted in a simple block sliding down an inclined plane with friction subject to the gravity acceleration. The goal of this test is to compare the angle at which the friction force can't hold the block anymore and it starts sliding downhill with the theoretical one, which is known. For a plane with a coefficient of friction with value $\mu = 0.5$ the limit inclination angle for the floor is:

$$\theta = atan(\mu) = 26.5650^{\circ} \tag{5.1}$$

The mass of the block is $m = 1\,\text{kg}$ and its dimensions are $1\,\text{m}$ by $0.5\,\text{m}$ by $0.5\,\text{m}$. Its longest side is oriented to match the sliding direction.

Furthermore, a 4 seconds simulation with a plane inclination of 30° was run in order to ensure sliding and compare the centroid trajectory, its velocity and the angle deviation of three local vectors initially aligned with the global x, y and z vectors. The trajectory error was measured as the distance from the block centroid to the floor sagittal plane. Under ideal conditions, this distance should be constantly zero, and so should be the three angles deviation with respect to the three global axis. The axis configuration is the following: the x axis points

forward and matches sliding direction, the y axis points upward and the z axis to the right of the block. The contact parameters can be seen in Table 5.6.

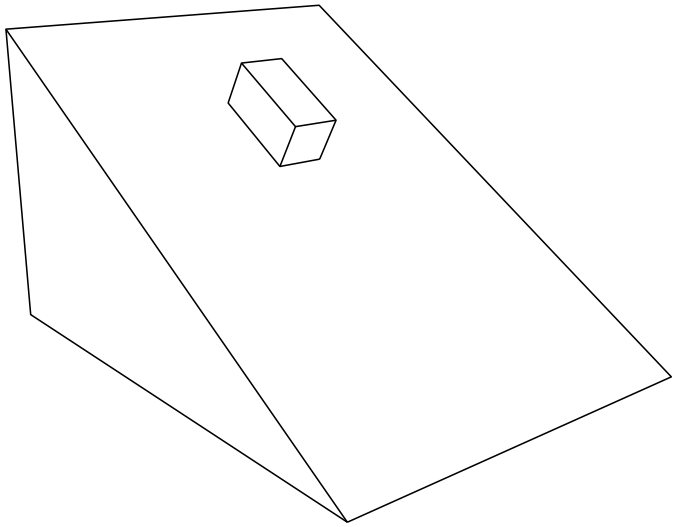

Figure 5.1: Block sliding on plane

5.1.2. Results

5.1.2.1. Mesh

The mesh model for this test is the simplest triangular mesh possible with two triangles per face (Figure 5.2). For the floor, another bigger block has been used with an equivalent triangle mesh.

The angle at which the sliding started was 26°. Compared with the theoretic value of 26.56°, this accounts for a 97.87% accuracy. The trajectory, velocity and angular errors can be seen in Figure 5.3, Figure 5.4 and Figure 5.5. The trajectory error remained under 5×10^{-6} during the 4 seconds simulation despite it seems to show a quadratic growth. The velocity evolution matches the theoretical one to a high degree of resemblance, and the biggest angular error, which corresponds with the pitch angle, was -6×10^{-5}. These results denote a very good correlation to the ideal performance in terms of accuracy.

It's important to take into account the effects of the initial position: in the theoretical solution, there is no inter-penetration between the block and the floor, but in the simulation a perfectly tangent, non-indentation initial position was set, so there exists a transition phase for the forces stabilization that lasts about 0.2 seconds and can be perceived both in the angular deviation and trajectory error charts.

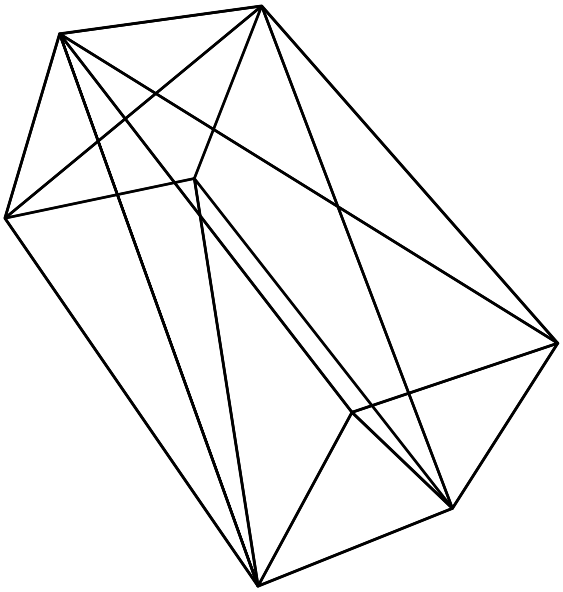

Figure 5.2: Sliding block mesh discretization

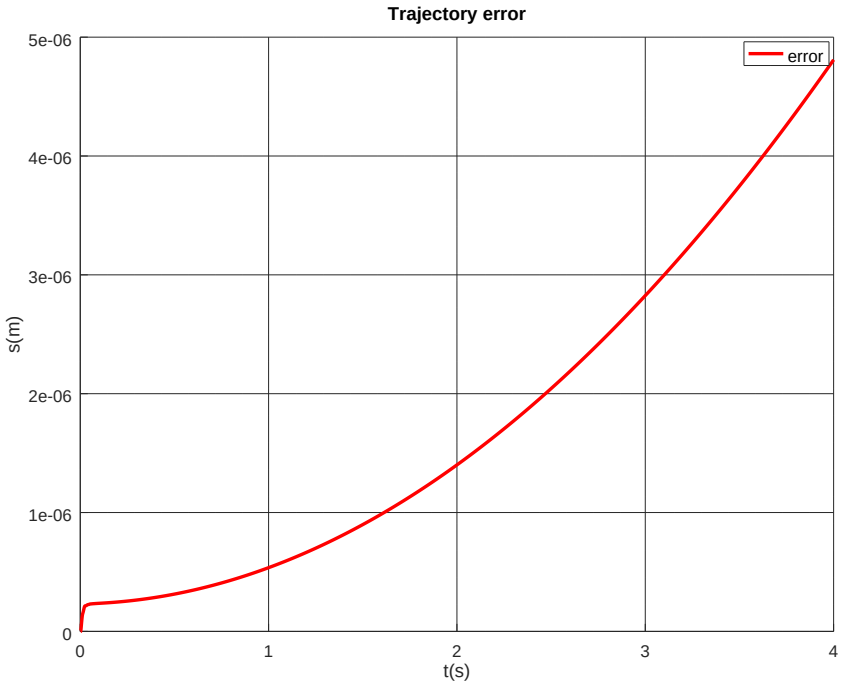

Figure 5.3: Sliding block (mesh): trajectory error

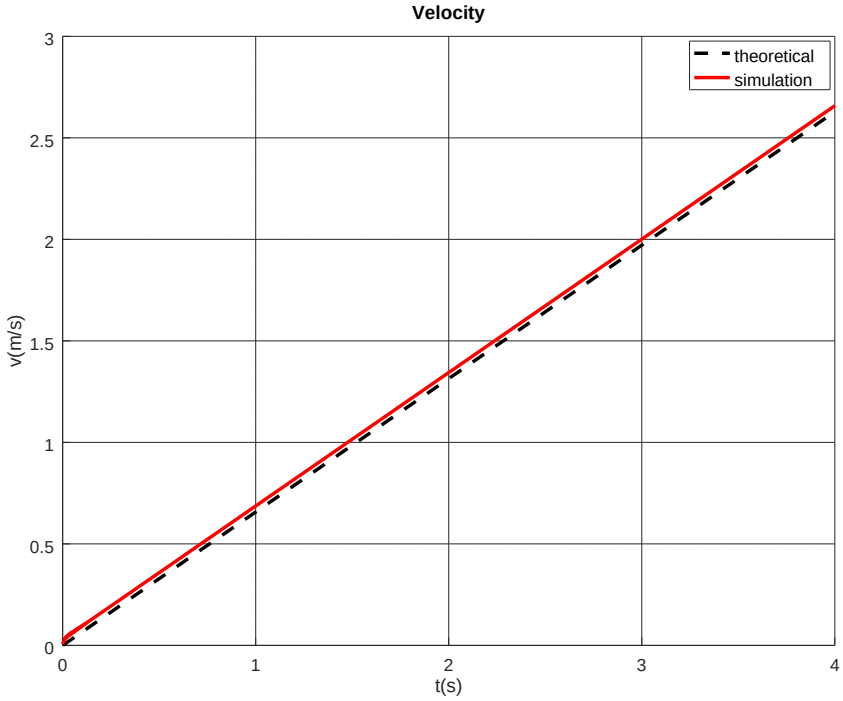

Figure 5.4: Sliding block (mesh): velocity error

5.1.2.2. Inner Sphere Tree

The IST model is an approximation of an object's geometry composed of spheres. The number of spheres selection poses a trade-off between the accuracy of the model and the computation cost. A higher number of spheres yields a more accurate object representation, but also means that the simulation execution time will be larger. As a rule of thumb, ISTs with in the range of 5000 to 20 000 spheres usually produce a fair enough representation for the objects used in this work. For this test, two ISTs of 12 215 and 15 423 spheres were used for the simulation's main body and the floor (Figure 5.6 and Figure 5.7).

In contrast to the previous mesh model, where only one force is calculated using data for the intersection, the IST model allows for the calculation of one force for every pair of colliding spheres. This means that the number of applied forces can be in the order of dozens, hundreds or even thousands. While this can be an advantage in the sense that replicates more faithfully the real interaction and adds up for the normal forces stabilization, it also creates some difficulties in the application of the tangential forces. The bristle model explained in Section 3.2.1 and used to solve the near-zero velocity tangential force problem, applied to such high number of forces, finds complications when trying to stop the block completely. When the sliding starts, a spring force is applied to every colliding sphere pair, and it is only

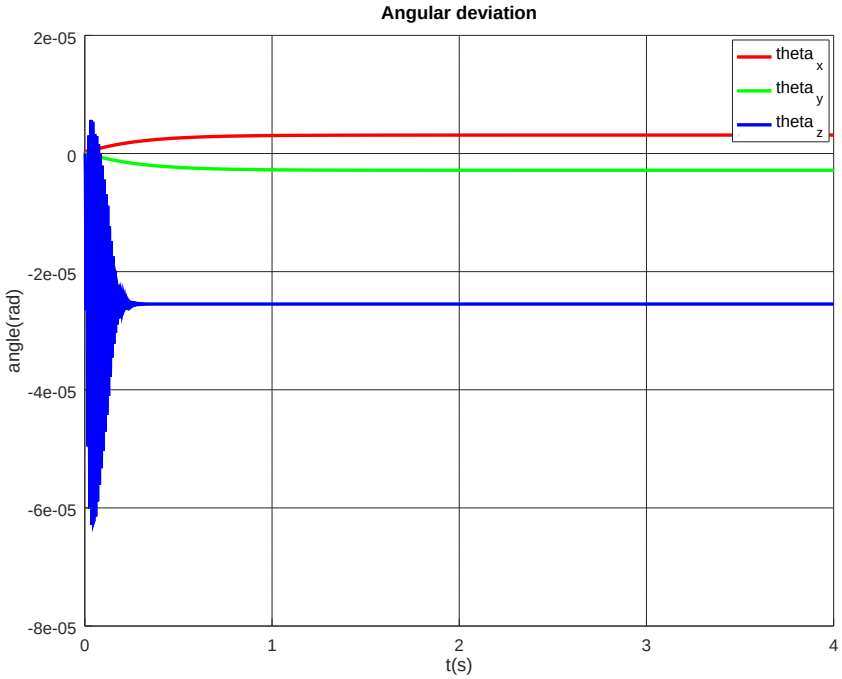

Figure 5.5: Sliding block (mesh): angular deviation

released when the tangential force surpasses the theoretical maximum.

$$F_{t_{\max}} = \mu F_n \tag{5.2}$$

As the sliding starts, lots of bristles are continuously triggered and released as the normal force is far from constant due to the roughness of the terrain made from spheres. This implies that the bristles can't stop the block completely, as a low residual sliding velocity always remains. This residual velocity depends on the plane inclination and its only perceptible to the eye from 22° on. Beyond 25° the block slides clearly. In Table 5.1 the inclination angles and its corresponding approximated residual velocities are represented for every order of magnitude.

Angle	Residual velocity (m/s)
10°	1×10^{-5}
22°	1×10^{-4}
23°	1×10^{-3}
25°	1×10^{-2}

Table 5.1: Sliding block IST: slope vs. residual velocity

The trajectory, velocity and angular errors can be seen in Figure 5.8, Figure 5.9 and Figure 5.10. In comparison with the mesh simulation, the IST model shows the

expected worse performance in terms of accuracy. The trajectory error moves in the range of centimeters, the velocity profile deviates greatly from the theoretical one and the angular deviation, while small, is significant if compared with the produced in the mesh simulation. The aforementioned initial transition phase can be increased here depending on the initial position, due to the greater fall in some positions where the first contacting spheres coincide with "valleys" in the other object.

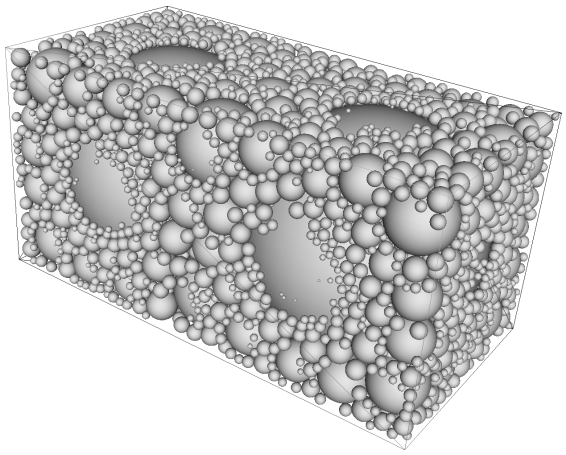

Figure 5.6: Sliding block sphere discretization (only the biggest 5000 spheres are shown)

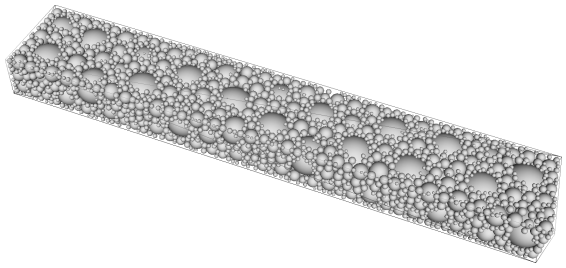

Figure 5.7: Sliding block floor sphere discretization (only the biggest 5000 spheres are shown)

5.1.2.3. Performance comparison

One of the main targets of this work is to be able to run these contact models in real-time environments such Virtual Reality simulations. Therefore, the execution time of every test was measured in order to assess the suitability of the models for their objective.

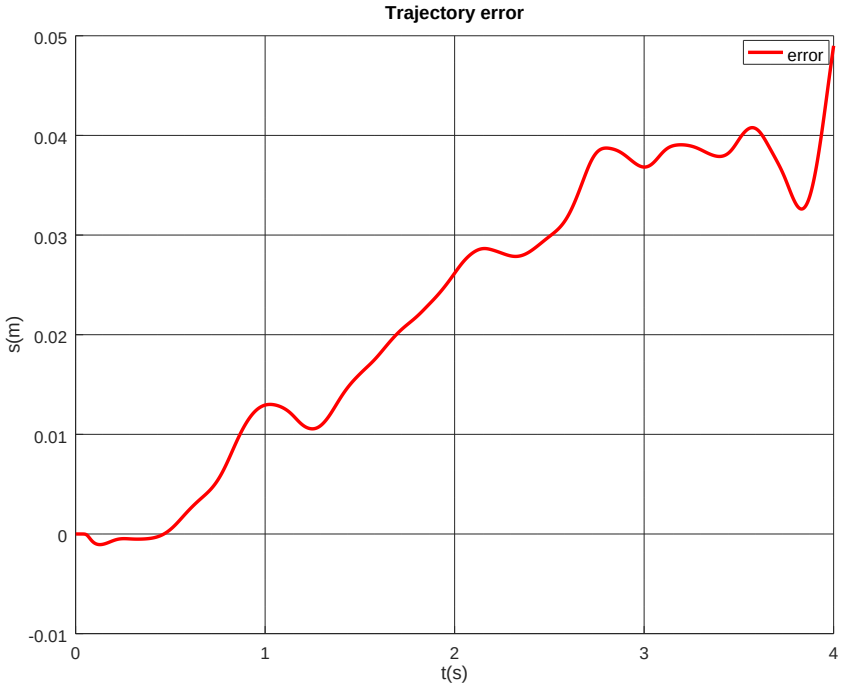

Figure 5.8: Sliding block (IST): trajectory error

The specs of the computer used to perform the simulations are summarized in Table 5.2.

Processor	Intel Core i7 950 @ 3GHz
RAM	6 GB DDR3
GPU	Nvidia Geforce GT 520
OS	elementary OS 0.4.1 Loki 64-bit
OS base	Ubuntu 16.04.5 LTS
Linux kernel version	4.9.1

Table 5.2: Computer specs

For this measurements, graphic output was disabled in order to reduce uncertainty and strictly measure the time spent on the simulation. It is also worth mentioning that no code optimization was made for any of the models. The measured time corresponds to the full simulation, from start to end, including object loading, preprocess routines execution and some datafile output. The results for this test can be seen in Table 5.3.

The real time ratio shows a fraction between the execution time and the simulated time. For a simulation to be run in real time, this number must be lower than 1. The mesh model simulation ran almost 19 times faster that the IST one, and

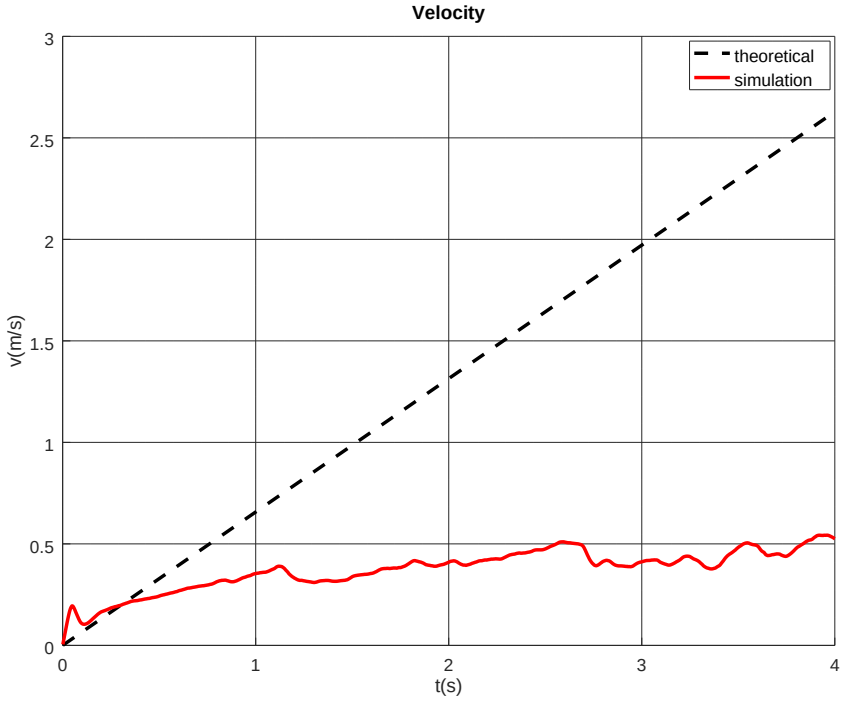

Figure 5.9: Sliding block (IST): velocity error

Model	Execution time (s)	Real time ratio
Mesh	1.362	0.340
IST	25.754	6.438

Table 5.3: Sliding block time execution performance

while the first beats the needs for real-time execution, the latter couldn't achieve this requirement.

5.2. Test 2: disk rotating on plane

5.2.1. Description

For this test, an uniform disk of radius $R = 0.25\,\mathrm{m}$, height $H = 0.05\,\mathrm{m}$ and mass $m = 1\,\mathrm{kg}$ is rotating over a plane at angular velocity $\omega_0 = 5\pi\,\mathrm{rad\,s^{-1}}$ ($2.5\,\mathrm{Hz}$). The coefficient of friction is $\mu = 0.6$ and no forces other than gravity interfere in the rotation.

The aim of this setup is to validate the spinning friction force model. The parameters selected for this validation are equivalent to the ones in the other tests: trajectory error, measured as the distance from the disk centroid to the

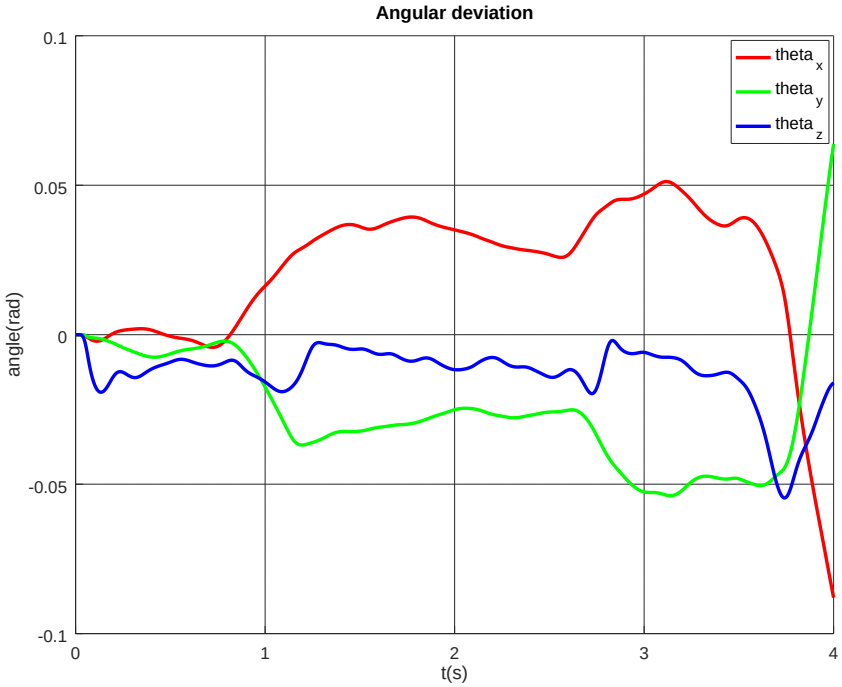

Figure 5.10: Sliding block (IST): angular deviation

origin, angular velocity profile as compared to the theoretical one, velocity error and angular deviation. In this case, the centroid should in theory not move during the simulation, and therefore the centroid distance to origin should be zero and its velocity null. The angular deviation is measured as the angle between the disk axis and the vertical axis, and its theoretical value is also zero.

To calculate the theoretical stop time for a spinning uniform disk, consider an infinitesimal thickness disk ring with dr radial dimension and dm mass like the one in Figure 5.12.

The uniformity of both the full disk and the infinitesimal ring enforces a constant relation between area and mass:

$$\frac{dm}{M} = \frac{\pi(r+dr)^2 - \pi r^2}{\pi R^2} \tag{5.3}$$

$$dm = \frac{2M}{R^2} r dr \tag{5.4}$$

The braking torque is provided by the tangential force, so

$$d\tau = rF_t = r\mu F_n = r\mu g dm \tag{5.5}$$

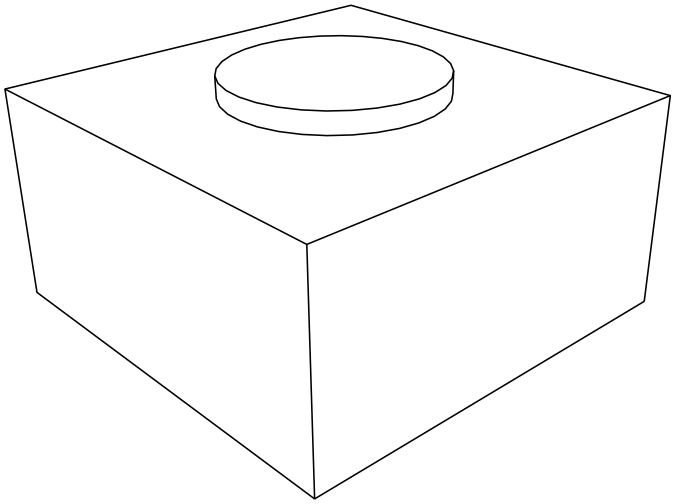

Figure 5.11: Disk rotating on plane

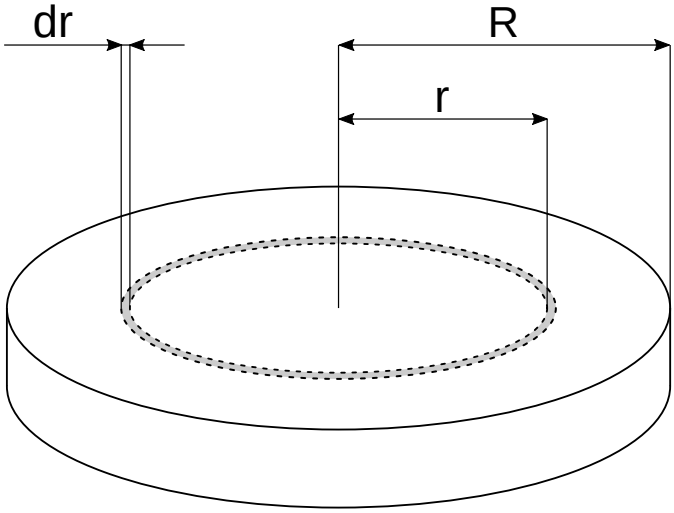

Figure 5.12: Uniform disk and infinitesimal thickness ring

Where F_t is the tangential force, F_n is the normal force, μ the friction coefficient and g the gravity constant.

Substituting 5.4 into 5.5:

$$d\tau = \frac{2M\mu g}{R^2} r^2 dr \tag{5.6}$$

$$\tau = \frac{2M\mu g}{R^2} \int_0^R r^2 dr \tag{5.7}$$

$$\tau = \frac{2}{3} M \mu g R \tag{5.8}$$

Which gives us the total exerted torque, that can also be calculated as:

$$\tau = I\alpha \tag{5.9}$$

Where I is the moment of inertia and α is the acceleration. For a uniform disk rotating over its axis, we have

$$I_{disk} = \frac{1}{2} M R^2 \tag{5.10}$$

Introducing Equation (5.8) into Equation (5.9):

$$\alpha = -\frac{\tau}{I} = -\frac{4\mu g}{3R} \tag{5.11}$$

Finally, the time it takes the disk to stop can be obtained using the expression for the angular velocity in an uniformly accelerated circular motion:

$$\omega = \omega_0 + \alpha t \tag{5.12}$$

$$t = -\frac{\omega_0}{\alpha} \tag{5.13}$$

$$t = \frac{3R\omega_0}{4\mu g} \tag{5.14}$$

With the given parameters, the friction generated by the gravity force should then stop the disk completely at $t = 0.5\,\mathrm{s}$

5.2.2. Results

5.2.2.1. Mesh

The mesh discretization for the rotating disk can be seen in Figure 5.13.

In an ideal situation, the rotating disk friction would linearly decrease its angular velocity until its full stop at 0.5 s. In Figure 5.14 we can see that this is not the case and that this braking process is more similar to a logarithmic decrease, followed by a full stop at around 1 s. This effect is due to the Gonthier spinning friction: while in the theoretical model the friction is constant, producing a lineal deceleration, the Gonthier spinning friction depends on the angular velocity, and therefore diminishes with time. Overall, the braking response is smooth but the braking time is inaccurate.

The maximum trajectory error was 1×10^{-4} (Figure 5.15) during the initial transition phase and stabilized at 5×10^{-5}. It is worth mentioning that for this simulation the sliding tangential force was deactivated due to integration problems. High angular velocities can complicate the integration when using vector coordinates instead of angles for the modelization. Despite it would be better for this test to use the spinning angle as a coordinate, the same modelization as the other test was intentionally used in order to compare all kind of situations in the same general conditions.

The velocity error (Figure 5.16) is only minimally significant during the initial phase but goes rapidly to zero, its expected value, and the angular deviation evolves in a similar way.

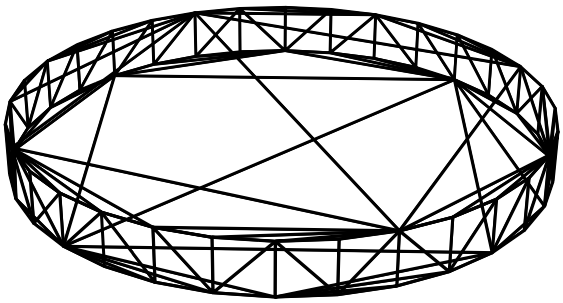

Figure 5.13: Rotating disk mesh discretization

5.2.2.2. Inner Sphere Tree

In the sphere test the disk was discretized with 11 212 spheres and the ground with 21 272 (Figure 5.19 and Figure 5.18). As seen in Figure 5.20, the angular velocity profile produced is very similar to a linear deceleration, despite the stop time was achieved 0.07 s earlier than the theoretical model. The maximum trajectory error (Figure 5.21) was 9 mm, which is two orders of magnitude greater than the mesh error, the velocity error (Figure 5.22) moved in the order of 0.05 m/s and the axis angular deviation was kept under 0.0022 rads.

Overall, the IST angular velocity evolution was more similar to the theoretical one than the mesh model but with a slightly less smooth response. The simulation results show more noise and vibration due to the effect of the colliding spheres. More spheres could be added to the model reduce this effect at the expense of a more computational expensive simulation.

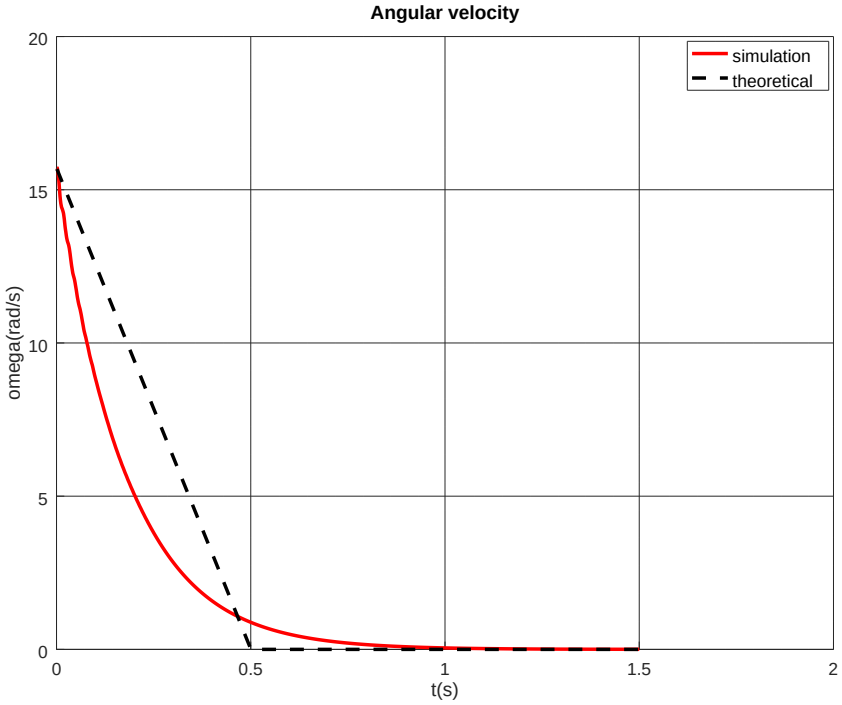

Figure 5.14: Rotating disk (mesh): angular velocity error

5.2.2.3. Performance comparison

Table 5.4 shows the time performance results for the rotating disk test. It has to be taken into account that as a result of this test duration depending directly on the stop time, both simulations lasted different times. The IST model braked significantly faster than the mesh model, so the simulation was stopped at $t = 0.5\,\mathrm{s}$ while the mesh simulation stopped at $t = 1.5\,\mathrm{s}$.

Model	Execution time (s)	Real time ratio
Mesh	1.973	1.315
IST	12.397	24.794

Table 5.4: Rotating disk time execution performance

For this test, none of the models met the real-time requirements. The fact that the spinning body made the integrator iterate multiple times until converging implied the simulation could not be run faster than real-time.

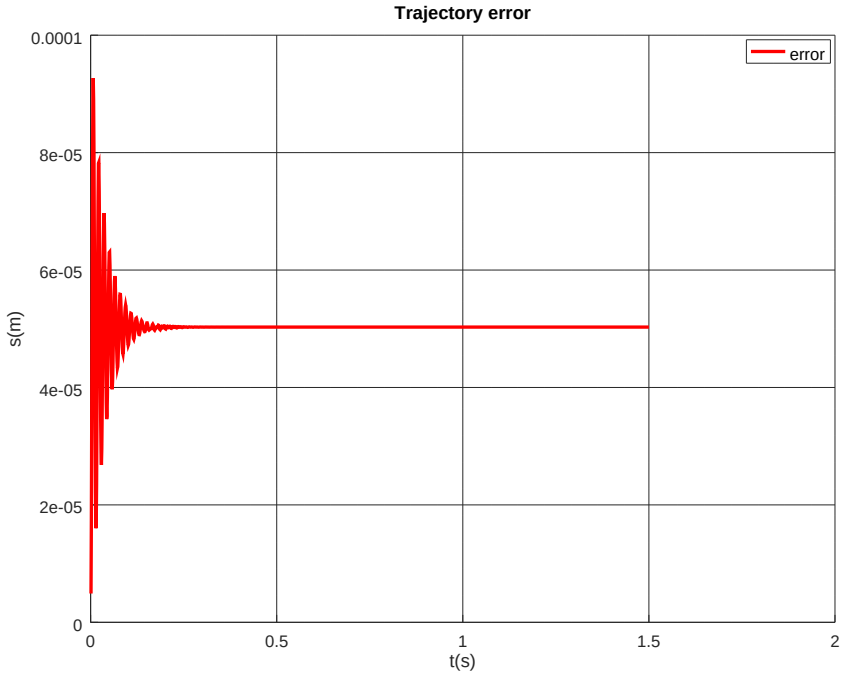

Figure 5.15: Rotating disk (mesh): trajectory error

5.3. Test 3: cylinder rolling on plane

5.3.1. Description

To check the quality of the rolling resistance model, one last test was designed. In this case, a cylinder of radius $R = 0.25\,\text{m}$, width $H = 1\,\text{m}$ and mass $M = 1\,\text{kg}$ rolls down a 15° inclined plane. The reason for the inclination angle selection was due to the fact that the IST model, with the current discretization, would not roll down planes less inclined. Very large inclinations like 30°, in the other hand, would produce an increasing frequency vibration in the mesh model, caused by the triangular faces of the cylinder's surface rotating at an increasing rotation rate. A simulation of 3.5 s was run and trajectory, velocity and angular errors were measured. As in previous tests, the trajectory error was measured as the distance from the cylinder centroid to the floor sagittal plane, which should be zero, the velocity profile was compared to the theoretical one and the angular deviation corresponds to the angle between the cylinder axis and the z-axis, that should also be null.

The simulation setup is depicted on Figure 5.24.

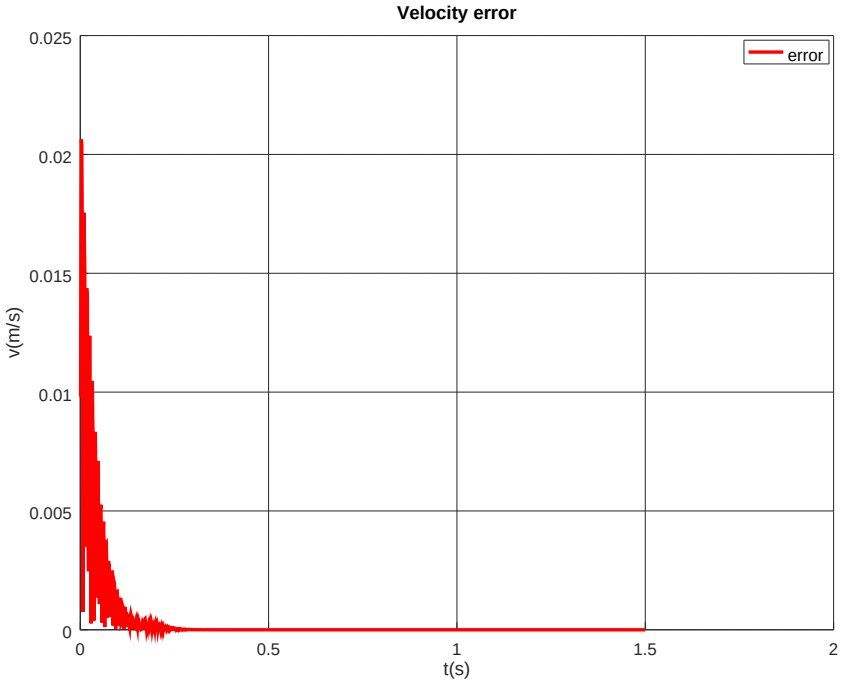

Figure 5.16: Rotating disk (mesh): velocity error

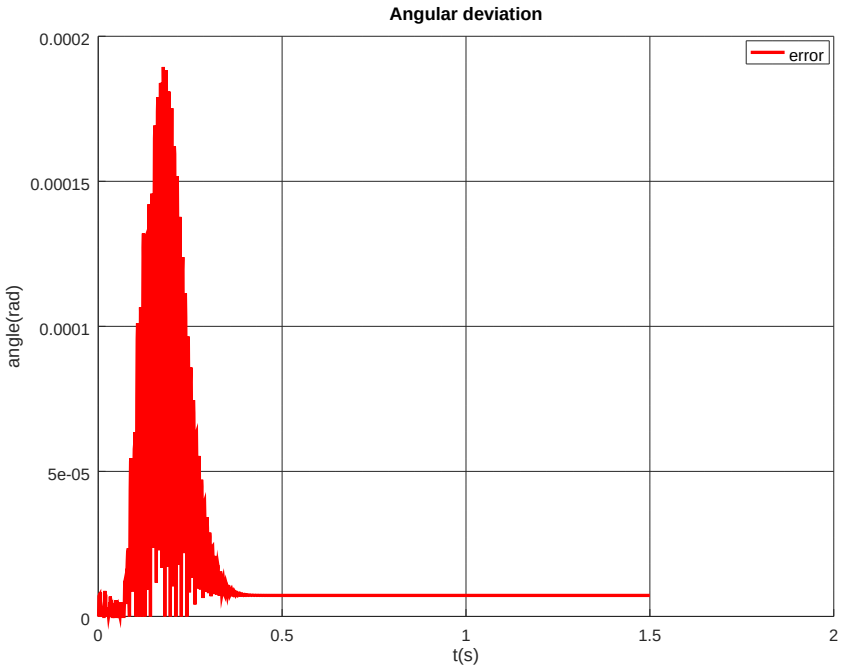

Figure 5.17: Rotating disk (mesh): angular deviation

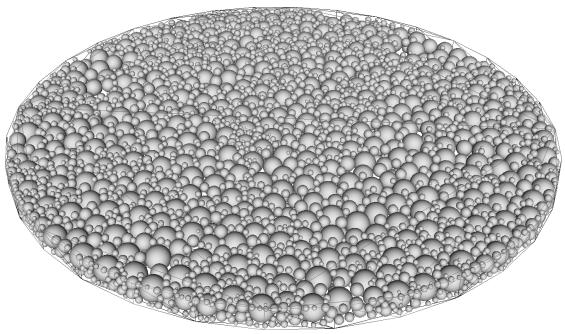

Figure 5.18: Rotating disk sphere discretization (only the biggest 5000 spheres are shown)

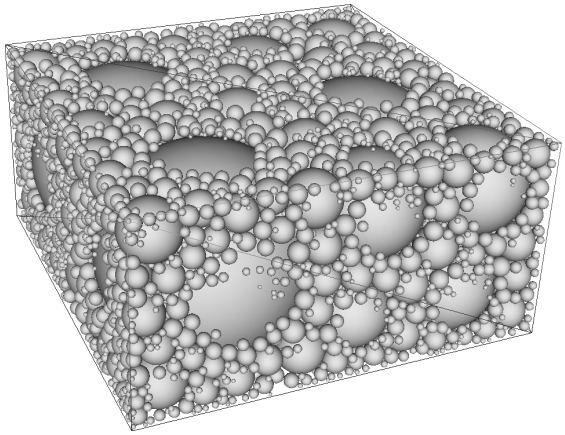

Figure 5.19: Rotating disk floor sphere discretization (only the biggest 5000 spheres are shown)

5.3.2. Results

5.3.2.1. Mesh

The mesh discretization for the rolling cylinder can be seen in Figure 5.25. Figure 5.26, Figure 5.27 and Figure 5.28 show the results.

Similarly to the first test, the trajectory error value remained very low (under 6×10^{-10}), but it shows an exponential evolution that starts to slow towards the end of the simulation. The velocity profile during the first second almost matches the theoretical solution, and from that point on, it starts drifting slowly until it reaches a maximum error of 8 % at the end. Nevertheless, the cylinder axis remained perfectly aligned with its original direction.

In general, the simulation appeared smooth and correct to the eye, but as stated before, great plane inclinations would induce increasing frequency noises in

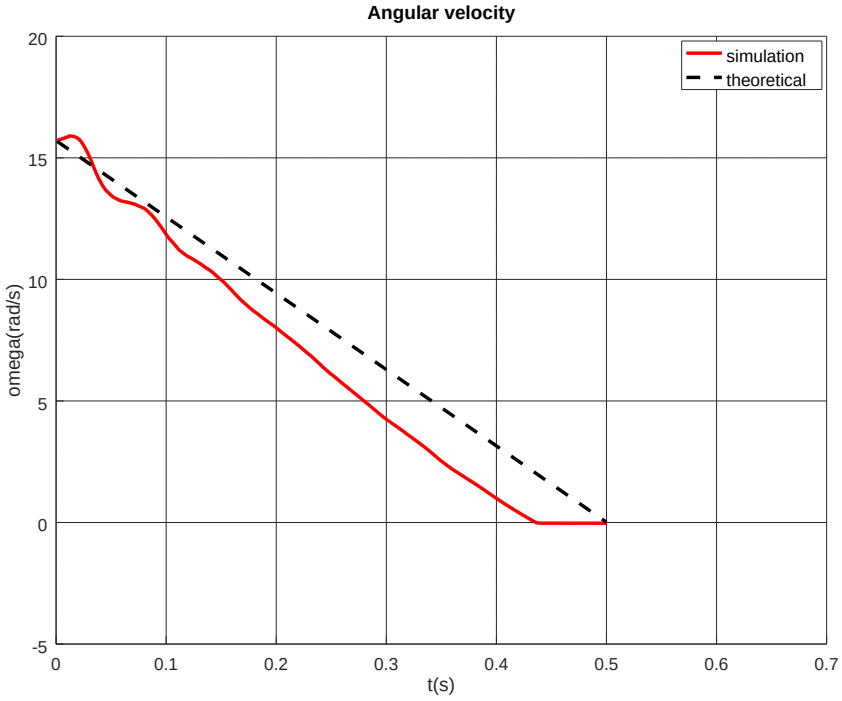

Figure 5.20: Rotating disk (IST): angular velocity error

the results.

5.3.2.2. Inner Sphere Trees

The mesh discretization for the rolling cylinder can be seen in Figure 5.29. It consists of 8849 spheres. For the ground model a change in the discretization strategy was needed due to highly incorrect trajectories in the simulations.

Using the ProtoSphere generated models with arbitrarily positioned spheres, the cylinder would roll out of the edges of the ground at very early stages of the simulation, rendering it unusable. The non-symmetrical sphere modelization produced forces that deviated the cylinder, making it turn 90° until it fell off the edge. For this reason, a new homogeneous ground sphere model was created. In this model, a layer of 200×30 spheres were aligned to avoid this deviation effect. It can be seen in Figure 5.30. While it could be thought that the same reasoning should apply to the cylinder model, the fact that it has radial symmetry helps the ProtoSphere tool to position the spheres in a more homogeneous way, creating the two first big spheres aligned with the cylinder axis and driving the cylinder in a more rectilinear trajectory. This can be seen in Figure 5.31: the maximum trajectory error is 1 cm.

The velocity profile (Figure 5.32) remained a little lower than the theoretical

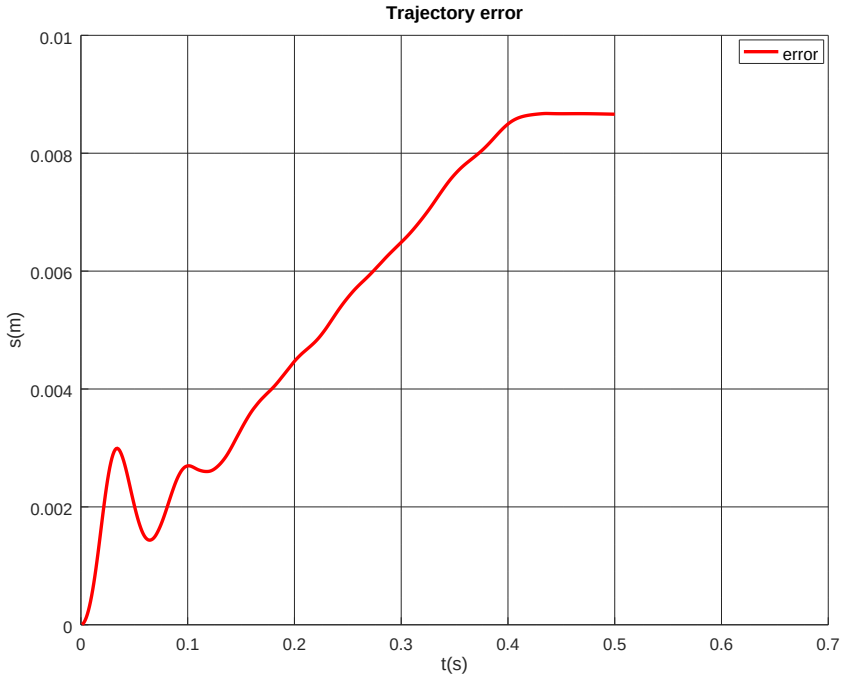

Figure 5.21: Rotating disk (IST): trajectory error

model, and the chart show the effect of the spheres collisions in the shape of "steps" in the first 2 seconds. As the rolling velocity increases, these are harder to perceive. The axis deviation (Figure 5.33) shows that the error remained under 1°.

5.3.2.3. Performance comparison

Table 5.5 shows the time performance results for the rolling cylinder test.

Model	Execution time (s)	Real time ratio
Mesh	1.893	0.540
IST	12.503	3.572

Table 5.5: Rolling cylinder time execution performance

Only the mesh model simulation, which ran 6.5 times faster that the IST model, could be executed faster than real-time.

5.4. Contact model parameters

The values used for the contact parameters in all the performed tests are collected in Table 5.6. These values were selected mainly as a function of the object

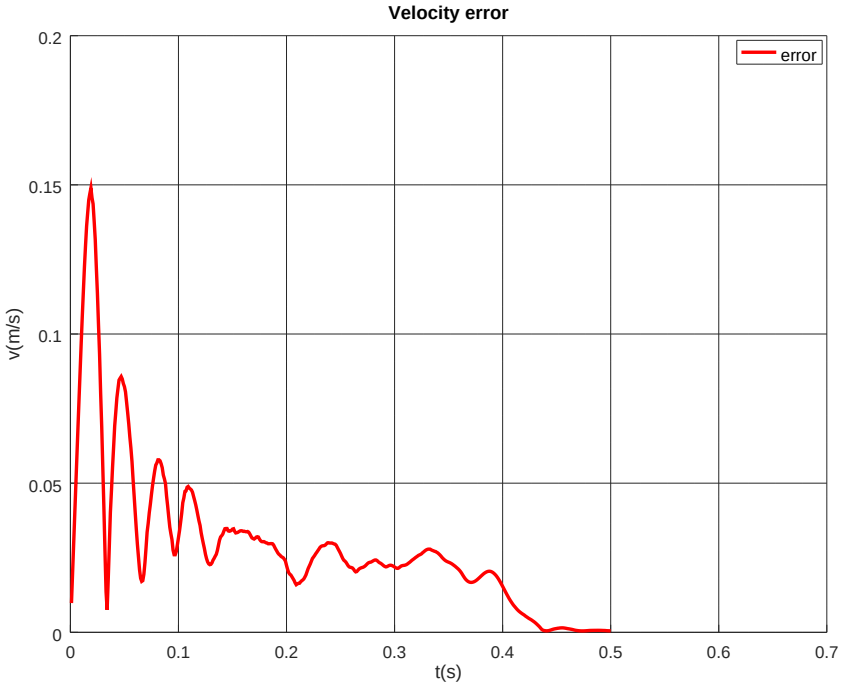

Figure 5.22: Rotating disk (IST): velocity error

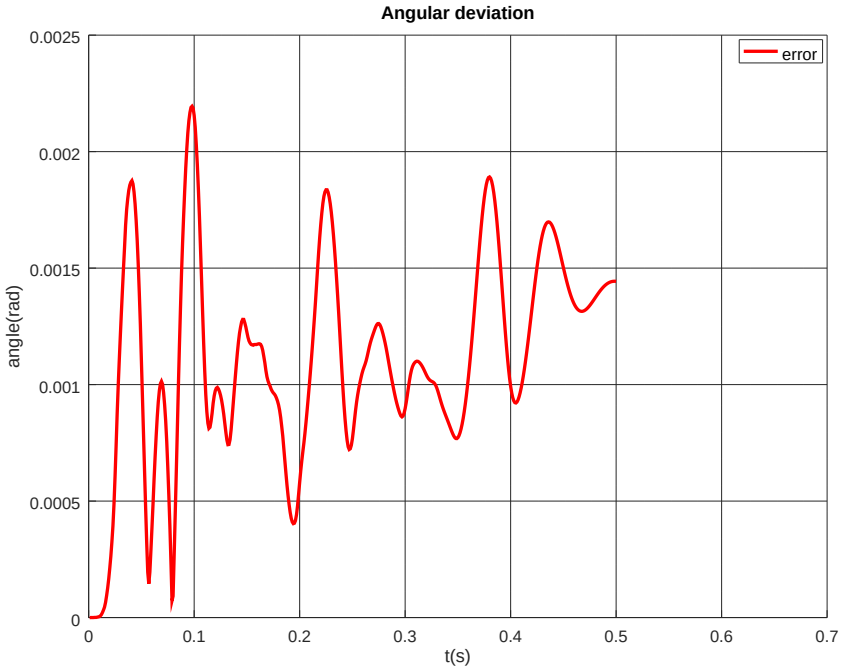

Figure 5.23: Rotating disk (IST): angular deviation

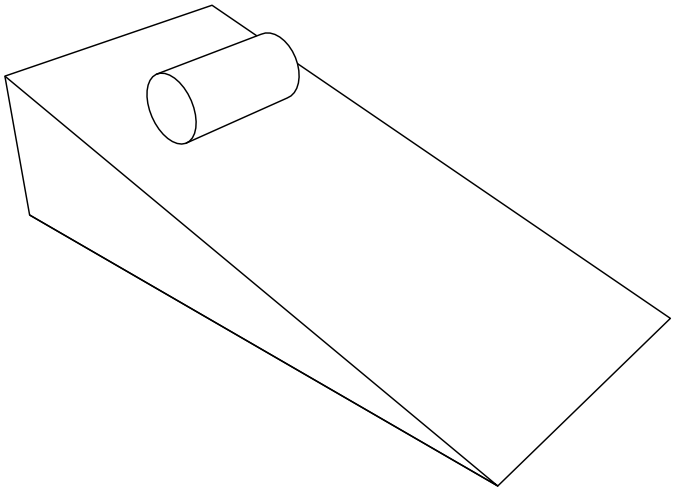

Figure 5.24: Cylinder rolling on plane

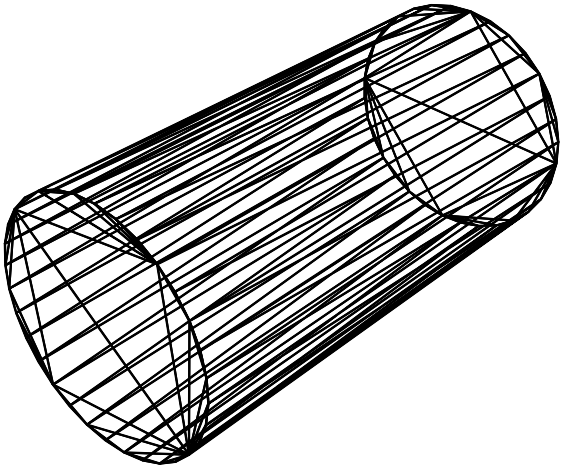

Figure 5.25: Rolling cylinder mesh discretization

masses and the time-step. Some of these parameters, like the stiffnesses for the normal model and the tangential one, had to be scaled for the IST simulations. The reason behind this scaling is the different number of forces in both methods: while in the mesh model we have only one collision and therefore one force, in the IST model dozens of collisions are processed at every time-step. Because of this, these stiffnesses need to be distributed along all the different contacts.

Also, the base stiffness value for the normal model in the IST method was selected so that the number of collisions was high enough to resemble a conforming contact. Otherwise, the collisions would be more similar to multiple simple contacts. Here 50 to 100 was enforced as the minimum number of collisions.

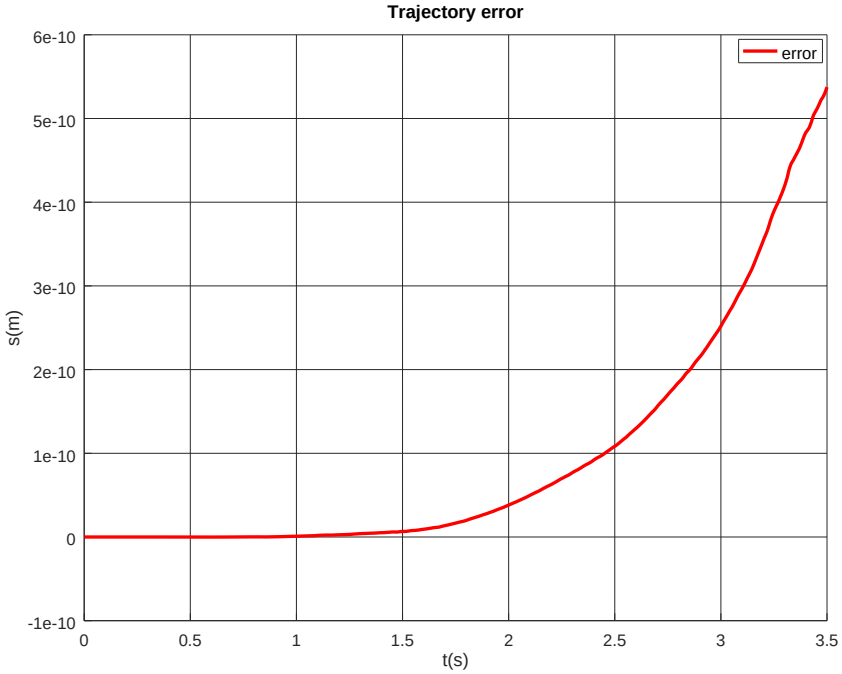

Figure 5.26: Rolling cylinder (mesh): trajectory error

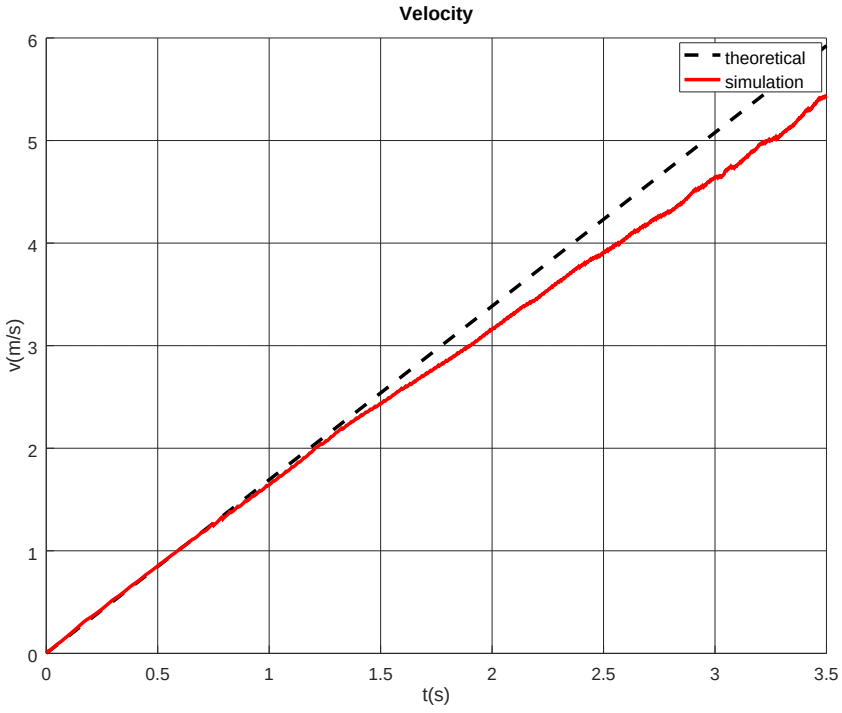

Figure 5.27: Rolling cylinder (mesh): velocity error

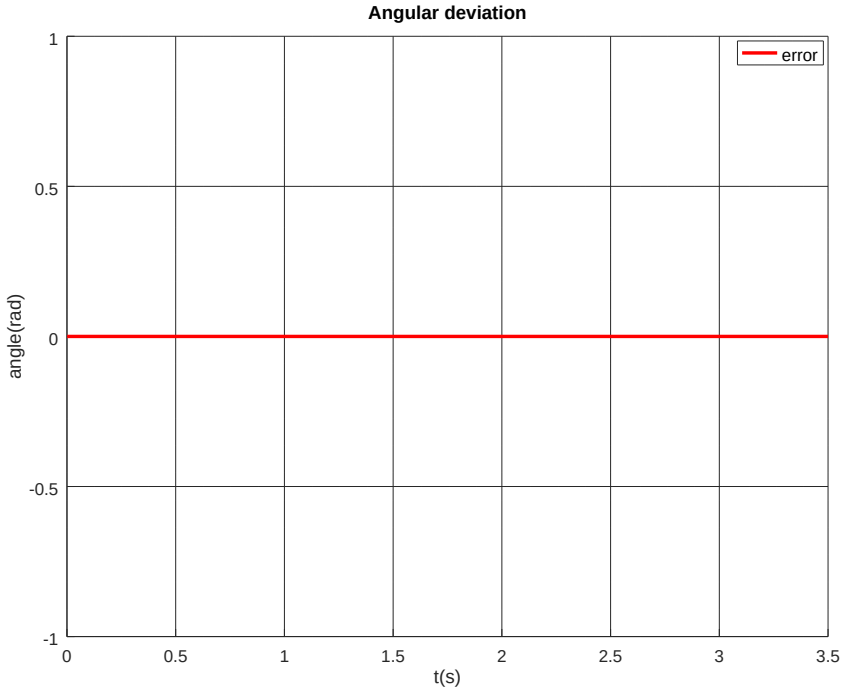

Figure 5.28: Rolling cylinder (mesh): angular deviation

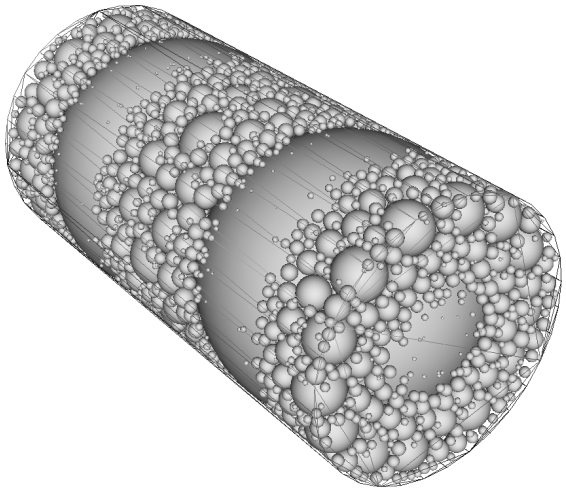

Figure 5.29: Rolling cylinder sphere discretization (only the biggest 5000 spheres are shown)

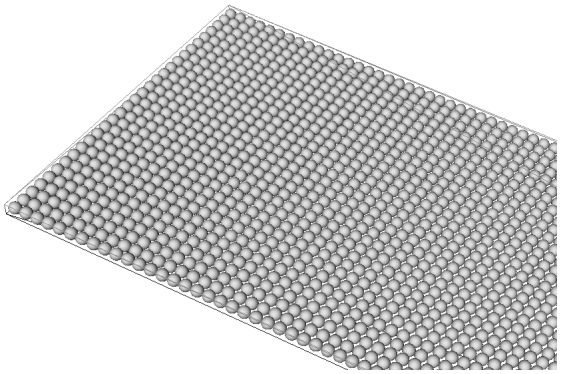

Figure 5.30: Rolling cylinder floor sphere discretization

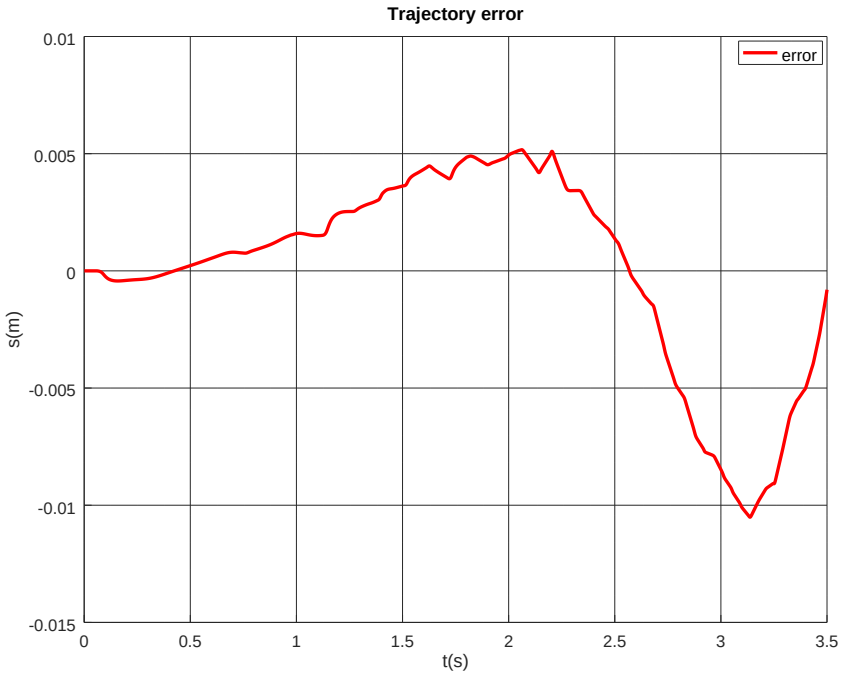

Figure 5.31: Rolling cylinder (IST): trajectory error

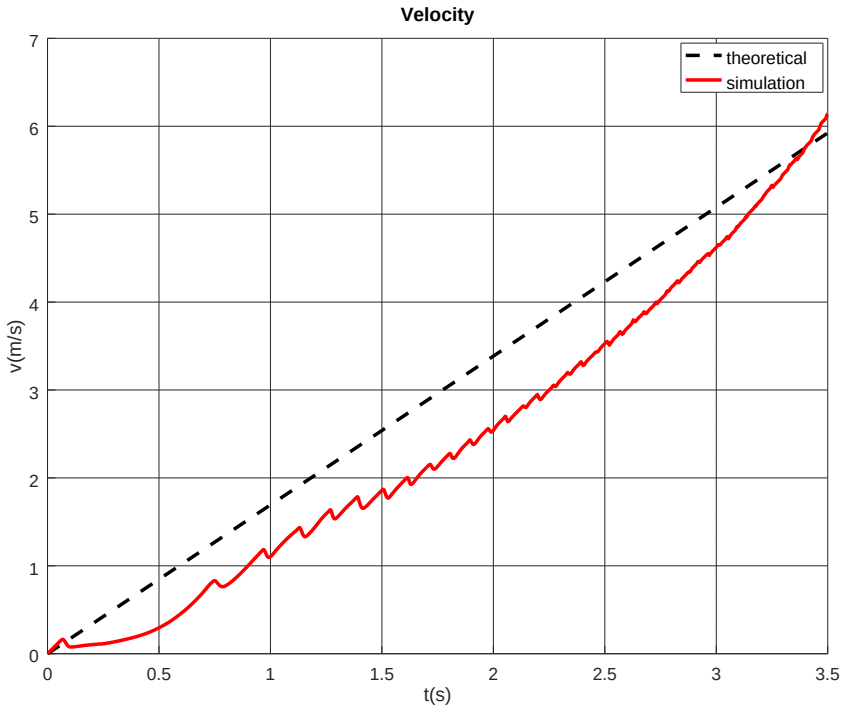

Figure 5.32: Rolling cylinder (IST): velocity error

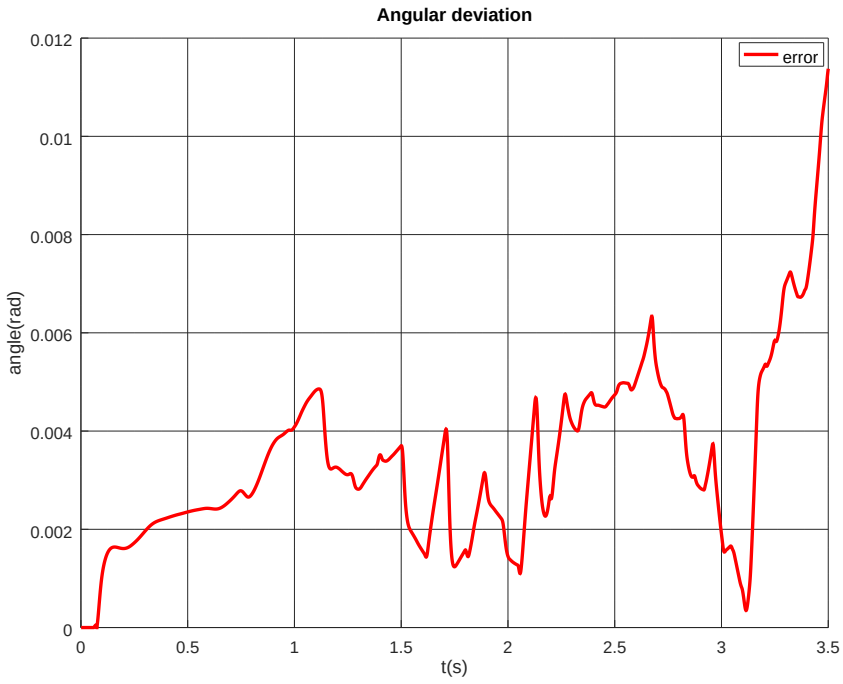

Figure 5.33: Rolling cylinder (IST): angular deviation

Parameter		Mesh			IST		
		Sliding block	Rotating disk	Rolling cylinder	Sliding block	Rotating disk	Rolling cylinder
Normal	μ	0.5	0.6		0.5	0.6	
	K	1×10^6			$1 \times 10^7 \frac{m}{n_{collisions}}$	$1 \times 10^9 \frac{m}{n_{collisions}}$	$1 \times 10^7 \frac{m}{n_{collisions}}$
	h	5					
Tangential	μ_{dyn}	0.5					
	μ_{sta}	μ_{dyn}					
	μ_{visc}	0					
	v_{stick}	$5 \times 10^{-3} \mu_{dyn} g$					
	k_{stick}	2.5×10^{-5}			$2.5 \times 10^{-5} \frac{m}{n_{collisions}}$		
	c_{stick}	$2\sqrt{k_{stick}}$					
	$p_0 updcoef$	0.8					

Table 5.6: Contact model parameters

Chapter 6

Conclusions and future work

6.1. Conclusions

This thesis aims to assess the feasibility of using volumetric force models in conforming contact situations while running in real-time environments, and to research the accuracy of these methods.

- In Chapter 2, an efficient and robust combination of multibody formulation, coordinates and integration method was described. This algorithm is fast enough to be used in interactive simulations, while remaining sufficient flexible to be able to enlarge or shrink the number of simulated bodies without high penalties.

- Chapter 3 described a volumetric contact model, including the computation of normal and friction forces, that is suitable to be used at interactive rates, yet producing realistic and accurate force responses.

- Chapter 4 discussed an algorithm aimed at determining the presence of contacts between the bodies in the simulation, based on their geometrical properties. The *far* and *near* detection stages were discussed, and two collision detection models were reviewed in order to evaluate both surface-defined and volume-defined object representations: one based in superficial triangular meshes combined with a Binary Space Partitioning tree (BSP) and other based in volumetric sphere fillings used along an Inner Sphere Tree (IST) hierarchy.

- Chapter 5 compared the performance of the two collision detection methods discussed in Chapter 4 in combination with the volumetric force model described in Chapter 3 in terms of both accuracy and execution speed.

The alliance between the multibody system formulation, the volumetric contact model and the collision detection method made it possible to develop an algorithm capable of simulating conforming contacts in real-time, therefore suitable for being used in interactive environments like virtual assembly simulators. The equations of motion were expressed in an Index-3 Augmented Lagrangian formulation and integrated with a fixed time-step of 1 millisecond by means of the trapezoidal rule. A Gonthier volumetric contact model was implemented in order to compute normal forces coupled with a tangential force model for describing friction phenomena. Two different collision detection algorithms, one based in triangular mesh surfaces and the other in volumetric filling spheres approximations, were employed to calculate the intersection properties during collisions to feed the needed input to the Gonthier model. For the mesh model, a collision detection library (LIMCODE) was written. This library calculates mesh-mesh intersections and was developed with results coherence in mind.

Three different tests were designed and implemented to investigate the strengths and weaknesses of the two approaches in the calculation of the intersection volumes. These tests were aimed to validate the accuracy of the force model in all kind of common contact situations that are present in virtual assembly simulations, and each of them was focused on one of the possible relative motions between two colliding bodies: sliding, rolling and spinning.

On top of our multibody dynamics library (MBSLIM) and our graphics collision detection libraries (MBSMODEL, LIMCODE), an export library was implemented (MBSDEBUG) in C++ in order to debug and visualize collisions, the clipping and intersection reconstruction process, and identify possible pitfalls for these algorithms. This library exports graphical data in EnSight6 format which can be opened with Paraview.

After evaluating the results of the tests, the following conclusions are drawn:

- The mesh collision detection model is able to produce real-time, accurate results in most cases, as expected. The fact of having an almost exact (exact in planar surfaces) representation of the volume intersection is clearly key in the calculation of the parameters needed for the force model.

  On the other hand, this model calculates just one total force. To calculate the direction of application of that force, a planar-contour contact must be assumed in order to approximate a contact plane and its normal, rendering the model less general. This problem and a possible workaround to solve it is explained with more detail in Section 6.2.6

- The IST model implementation, despite not being able to run in real-time or yield highly realistic simulations, is simpler and more general: it reduces a complex volumetric collision to a number of simple sphere-sphere contacts

and consequently to a number of forces, thus resembling more a force distribution.

Despite of it, the approximation made when using this model seems too rough to accurately represent virtual-assembly-grade collisions, at least in a realistic way.

- In the performed tests, with the selected number of spheres for the IST model, the mesh model ran up to almost 20 times faster. Probably, the IST evaluation would be faster with lower sphere counts, but this unfortunately means to lose accuracy and realism in the simulation, which with 10 000 to 15 000 is already not enough to meet the requirements, as seen previously. Furthermore, the need to completely fill objects with spheres means that, in objects with low surface-to-volume ratio, like the ones used in the tests, most of the spheres are not involved in collisions as they are internal spheres that never interact with the other body spheres due to being located deeper than the maximum penetration. This causes a big inefficiency from a memory point of view, as all the information of the spheres is loaded despite not being used in its majority. This is worsened by the fact that the stiction model needs to keep track of the collisions state in previous time-steps, thus incrementing the amount of information that must be saved for every sphere pair. This results in a time consuming list search that is run for every active sphere pair in every time-step.

The presented results are promising as a starting point for the development of real-time simulators with conforming contact requirements. However, different lines of research could help to improve both their efficiency and accuracy.

6.2. Future work

Several lines of research could pose interesting improvements over the actual status of this work.

6.2.1. Collision generalization

While the tests described in this thesis cover some situations that can arise in simple conforming contact conditions, new tests should be created in order to validate the performance of these methods in more complex situations, like a cube (or any other shape) inserted into a hole where it fits perfectly. This kind of tests involves multi-planar conforming contact and it would be interesting to study how a volumetric contact model performs in such situations, as this type

of collisions occurs frequently in virtual assembly environments. Furthermore, general collisions where several friction force types appear at the same time should be checked to ensure friction models do not interfere with each other.

Also, tests with many more objects than just two could be carried out to learn how the problem size affects the speed of computation. It is expected for a virtual reality simulation to have multiple objects interacting, so a good performance with a high number of meshes must be ensured.

Finally, an interactive virtual assembly environment to test all kind of collisions would be the ideal testbed to validate the applicability of this kind of methods for production.

6.2.2. Optimizations

Besides the purely programming optimizations that can always be done in order to boost computation speeds, some changes can also improve simulation times.

- **Using larger time-steps:** currently a fixed time-step of 1 millisecond is being used, but this could be increased to 3, 5 or even 10 milliseconds as long as the accuracy doesn't get affected. Visually, most users won't perceive frame-rate refreshes greater that 60 Frames Per Second (FPS), so increasing the time-step could potentially allow for the inclusion of more interacting objects in the simulation while keeping a good enough visual feedback. Nonetheless, if haptic devices are connected to the simulation, this has to be handled more carefully since the optimal refresh rate for these machines is 1KHz.

- **Calling contact detection one time per time-step:** in some time-steps, specially those when high speed objects are under collision or when complex contact situations are taking place, the integrator has to iterate several times before converging to a solution. This implies very small displacements of the objects which usually means very little or no change in their colliding meshes. Despite this fact, collision detection is being called unconditionally at each iteration. Being the most CPU-time consuming task, getting rid of this extra calls could carry noticeable time savings.

6.2.3. Parallelization

The methods presented in this document perform all the computations in the same processing line, i.e. all the tasks are serialized one after the other. Current computing architectures, even domestic devices, have the capability of performing several tasks simultaneously. A process that performs all its computation in a serial order, has at most the chance to make use of a fraction of the computing

power of the processing unit. As the simulated system is enlarged, techniques like the ones presented in [38] can alleviate the problem, but they can become ineffective if there is a huge number of bodies interacting in the scene.

Subdivide an algorithm into sub-duties that can be computed in parallel is not a trivial task, since synchronization stages between independent processes have a non-negligible computing cost. For each kind of problem, a satisfactory parallelization scheme must be found that minimizes the information transfers — and thus the need for synchronization steps— between processes. In [28] warnings are stated about the high penalty that small or medium-sized multibody problems incur into when trying to parallelize: usually the bottlenecks are the computation of the Jacobian of the constraints and the solving of the final linear system of equations.

An interesting research line consists in dividing the multibody system on several sub-mechanisms having their own processes, and interacting by exchanging reaction forces. Synchronization costs can be avoided using *Inter-Process Communication*s for data passing between the processes.

6.2.4. Time critical IST collision detection

Time-critical collision detection algorithms would imply to adjust the volumetric intersection properties calculations to a predefined time budget. Some early work was done in [58] in order to implement this feature into the CollDet library. This approach would mean that during the IST traversal, an approximate intersection volume would be returned instead of the exact one if the integration step is running out of time. This behavior ensures to be capable of always running at real-time speeds at the cost of losing accuracy in the most complex collisions. Using this modified algorithm could be beneficial for keeping an interactive frame rate, at least for human-in-the loop or hardware-in-the-loop simulations.

6.2.5. Multiple contact stiction

The IST model was not able to completely stop the sliding block in Section 5.1.2.2. It seems that the stiction model used may have some problems when dealing with dozens or hundreds of contacts simultaneously. More research is needed in order to clarify this behavior.

6.2.6. Arbitrary shape decomposition

Force models always need a point and a direction of application in order to be introduced into the simulation. This poses a setback for volumetric models like the mesh model presented in this work: an intersection volume does not have such

direction, so to circumvent this problem and calculate a normal vector, planar-contour contacts are assumed and the collision contour between the bodies is used for approximating a plane through the least squares fitting method. The normal of this plane is then used as the direction of the normal force, but this procedure relegates the model to the aforementioned planar-contour contact situations, as multi-planar intersections are not assimilable to a single plane and thus do not satisfy Gonthier model requirements.

The IST model doesn't suffer from this exact problem, as every sphere pair contact is treated separately and its normal is easily calculated as the director vector of the line along both sphere centers. In return, many of these normals are inaccurate and often need some kind of filtering to avoid strange behaviors. Examples of this would be the collision between a sphere from body A that has penetrated completely the outermost sphere shell in body B and generates an attraction force when the two bodies start separating, or forces with a high tangential component due to the spheres colliding sideways.

For the mesh model to be able to confront non-planar-contour collisions (e.g. a cube inserted in a cubic hole with its same dimensions), an arbitrary shape decomposition could be applied. Bodies would be preprocessed and split into a set of convex shapes, and every piece would be used to calculate a force through the presented mesh model. This way every part in contact would generate an individual convex-convex collision suitable to feed the model.

Bibliography

[1] E. Bayo and R. Ledesma. "Augmented Lagrangian and mass–orthogonal projection methods for constrained multibody dynamics". In: *Nonlinear Dynamics* 9.1-2 (1996), pp. 113–130.

[2] E. Bayo, J. Garcia De Jalon, and M. A. Serna. "A modified lagrangian formulation for the dynamic analysis of constrained mechanical systems". In: *Computer Methods in Applied Mechanics and Engineering* 71.2 (Nov. 1, 1988), pp. 183–195.

[3] E. Bayo and R. Ledesma. "Augmented Lagrangian and mass-orthogonal projection methods for constrained multibody dynamics". In: *Nonlinear Dynamics* 9 (Feb. 1, 1996).

[4] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2004.

[5] G. Bernstein and D. Fussell. "Fast and Exact and Linear Booleans". In: *Eurographics Symposium on Geometry Processing 2009 Volume 28 (2009), Number 5 Marc Alexa and Michael Kazhdan (Guest Editors)*. 2009.

[6] K. Bhalerao, S. Anderson, and J. Trinkle. "A Recursive hybrid time-stepping scheme for intermittent contact in multi-rigid-body dynamics". In: *Journal of Computational and Nonlinear Dynamics* 4.4 (2009), pp. 1–11.

[7] K. Brenan, S. Campbell, and L. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. New York: North-Holland, 1989.

[8] E. A. Butcher and D. J. Segalman. "Characterizing Damping and Restitution in Compliant Impacts via Modified K-V and Higher-Order Linear Viscoelastic Models". In: *Journal of Applied Mechanics* 67.4 (Feb. 10, 2000), pp. 831–834.

[9] J. Chung and G. Hulbert. "A time integration algorithm for structural dynamics with improved numerical dissipation: the generalized-alpha method". In: *ASME Journal of Applied Mechanics* 60 (1993), pp. 371–375.

[10] J. Cuadrado, R. Gutiérrez, M. Naya, and P. Morer. "A comparison in terms of accuracy and efficiency between a MBS dynamic formulation with stress analysis and a non–linear FEA code". In: *International Journal for Numerical Methods in Engineering* 51.9 (2001), pp. 1033–1052.

[11] J. Cuadrado, J. Cardenal, P. Morer, and E. Bayo. "Intelligent Simulation of Multibody Dynamics: Space–State and Descriptor Methods in Sequential and Parallel Computing Environments". In: *Multibody System Dynamics* 4.1 (2000), pp. 55–73.

[12] O. Devillers and S. Pion. *Efficient Exact Geometric Predicates for Delaunay Triangulations*. Tech. rep. RR-4351. INRIA, Jan. 2002.

[13] A. DiCarlo and A. Paoluzzi. "Fast Computation Of Inertia Through Affinely Extended Euler And Tensor". In: *Computer-Aided Design*. Vol. 2016-November. 2006, pp. 1145–1153.

[14] S. Djerassi. "Collision with friction; Part A: Newton's hypothesis". In: *Multibody System Dynamics* 21.1 (Feb. 1, 2009), p. 37.

[15] S. Djerassi. "Collision with friction; Part B: Poisson's and Stronge's hypotheses". In: *Multibody System Dynamics* 21.1 (Feb. 1, 2009), p. 55.

[16] D. Dopico. "Formulaciones semi-recursivas y de penalización para la dinámica en tiempo real de sistemas multicuerpo". PhD thesis. Universidade da Coruña, Oct. 2004.

[17] D. Dopico, A. Luaces, M. Gonzalez, and J. Cuadrado. "Dealing with multiple contacts in a human-in-the-loop application". In: *Multibody System Dynamics* 25.2 (Feb. 2011), pp. 167–183.

[18] P. Eberhard. "Computational Dynamics of Multibody Systems: History, Formalisms, and Applications". In: *European Journal of Mechanics* 25.4 (2006), pp. 566–594.

[19] D. Eberly. *Polyhedral Mass Properties (Revisited)*. 2002. URL: `https://www.geometrictools.com/Documentation/PolyhedralMassProperties.pdf` (visited on 07/12/2018).

[20] H. Edelsbrunner and E. P. Mücke. "Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms". In: *ACM Trans. Graph.* 9.1 (Jan. 1990), pp. 66–104.

[21] E. Eich-Soellner and C. Führer. *Numerical Methods in Multibody Dynamics*. Red. by L. Arkeryd, H. Engl, A. Fasano, R. M. M. Mattheij, et al. European Consortium for Mathematics in Industry. Wiesbaden: Vieweg+Teubner Verlag, 1998.

[22] C. Ericson. *Real Time Collision Detection*. Morgan Kaufmann, 2005.

[23] P. Flores, J. Ambrósio, J. C. P. Claro, and H. M. Lankarani. "Influence of the contact-impact force model on the dynamic response of multi-body systems". In: *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 220.1 (2006), pp. 21–34.

[24] P. Flores, R. Leine, and C. Glocker. "Modeling and analysis of planar rigid multibody systems with translational clearance joints based on the non-smooth dynamics approach". In: *Multibody System Dynamics* 23.2 (Feb. 1, 2010), pp. 165–190.

[25] P. Flores, J. Ambrósio, J. C. Claro, and H. M. Lankarani. *Kinematics and Dynamics of Multibody Systems with Imperfect Joints*. Vol. 34. Lecture Notes in Applied and Computational Mechanics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[26] Y. Gonthier, J. McPhee, C. Lange, and J. Piedboeuf. "A contact modeling method based on volumetric properties". In: *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. ASME, 2005, pp. 477–486.

[27] Y. Gonthier, J. McPhee, C. Lange, and J. Piedboeuf. "A regularized contact model with asymmetric damping and dwell-time dependent friction". In: *Multibody System Dynamics* 11.3 (2004), pp. 209–233.

[28] F. González, A. Luaces, U. Lugrís, and M. González. "Non–intrusive parallelization of multibody system dynamic simulations". In: *Computational Mechanics* 44.4 (2009), pp. 493–504.

[29] G. Guennebaud, B. Jacob, et al. *Eigen: a C++ Template Library for Linear Algebra: Matrices, Vectors, Numerical Solvers, and Related Algorithms*. July 2018. URL: `http://eigen.tuxfamily.org`.

[30] P. Guigue and O. Devillers. "Fast and Robust Triangle-Triangle Overlap Test Using Orientation Predicates". In: *Journal of Graphics Tools* 8 (Jan. 2003).

[31] E. Hairer and G. Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer- Verlag, Berlin Heidelberg, 1996.

[32] H. Hilber, T. Hughes, and R. Taylor. "Improved numerical dissipation for time integration algotithms in structural dynamics". In: *Earthquake Engineering and Structural Dynamics* 5 (1977), pp. 283–292.

[33] K. Hunt and E. Crossley. "Coefficient of restitution interpreted as damping in vibroimpact". In: *Journal of Applied Mechanics* (1975).

[34] K. A. Ismail and W. J. Stronge. "Impact of Viscoplastic Bodies: Dissipation and Restitution". In: *Journal of Applied Mechanics* 75.6 (Aug. 20, 2008), pp. 061011–061011–5.

[35] J. García de Jalón and E. Bayo. *Kinematic and dynamic simulation of multibody systems: the real time challenge*. Springer-Verlag, 1994.

[36] H. M. Lankarani and P. E. Nikravesh. "A Contact Force Model With Hysteresis Damping for Impact Analysis of Multibody Systems". In: *Journal of Mechanical Design* 112.3 (1990), p. 369.

[37] D. T. Lee and F. P. Preparata. "An Optimal Algorithm for Finding the Kernel of a Polygon". In: *J. ACM* 26.3 (July 1979), pp. 415–421.

[38] A. Luaces. "Contact and HiL Interaction in Multibody Based Machinery Simulators". PhD thesis. Universidade da Coruña, 2013.

[39] J. Lyness and C. Moler. "Numerical Differentiation of Analytic Functions". In: *SIAM Journal on Numerical Analysis* 4.2 (1967), pp. 202–210.

[40] P. Lötstedt. "Mechanical Systems of Rigid Bodies Subject to Unilateral Constraints". In: *SIAM Journal on Applied Mathematics* 42.2 (Apr. 1, 1982), pp. 281–296.

[41] M. Machado, P. Flores, and J. Ambrósio. "A lookup-table-based approach for spatial analysis of contact problems". In: *Journal of Computational and Nonlinear Dynamics* 9.4 (2014).

[42] B. Mirtich. "Fast and Accurate Computation of Polyhedral Mass Properties". In: *Journal of Graphics Tools* 1.2 (Feb. 1996), pp. 31–50.

[43] N. M. Newmark. "A method of computation for structural dynamics". In: *Journal of the Engineering Mechanics Division, ASCE* 85.EM3 (1959), pp. 67–94.

[44] P. E. Nikravesh. *Computer-aided analysis of mechanical systems*. Engelwood Cliffs, NJ, USA: Prentice-Hall, 1988.

[45] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag, 1999.

[46] N. V. Orlandea, D. A. Calahan, and M. A. Chace. "A sparsity-oriented approach to the dynamic analysis and design of mechanical systems – Part 1". In: *Journal of Engineering for Industry* 99.3 (1977), pp. 773–779.

[47] P. Panchekha, A. Sachez-Stern, J. R. Wilcox, and Z. Tatlock. "Automatically improving accuracy for floating point expressions". In: *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. Vol. 2015-June. 2015, pp. 1–11.

[48] R. Pastorino, F. Cosco, F. Naets, W. Desmet, and J. Cuadrado. "Hard real-time multibody simulations using ARM-based embedded systems". In: *Multibody System Dynamics* 37.1 (2016), pp. 127–143.

[49] N. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Jan. 2002.

[50] F. Pfeiffer and C. Glocker. *Multibody Dynamics with Unilateral Contacts*. CISM International Centre for Mechanical Sciences. Springer Vienna, 2000.

[51] H. Rahnejat. "Multi-body dynamics: historical evolution and application". In: *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science* 214.1 (2000), pp. 149–173.

[52] W. Schiehlen. "Multibody system dynamics: roots and perspectives". In: *Multibody system dynamics* 1.2 (1997), pp. 149–188.

[53] J. Shewchuk. "Adaptive precision floating-point arithmetic and fast robust geometric predicates". In: *Discrete and Computational Geometry* 18.3 (1997), pp. 305–363.

[54] W. Squire and G. Trapp. "Using Complex Variables to Estimate Derivatives of Real Functions". In: *SIAM Review* 40.1 (1998), pp. 110–112.

[55] W. J. Stronge. *Impact Mechanics*. Cambridge University Press, Mar. 25, 2004. 306 pp.

[56] K. Sugihara and M. Iri. "A Solid Modelling System Free from Topological Inconsistency". In: *J. Inf. Process.* 12.4 (Apr. 1990), pp. 380–393.

[57] W. C. Thibault. "Application of Binary Space Partitioning Trees to Geometric Modeling and Ray-Tracing". PhD thesis. School of Information and Computer Science Georgia Institute of Technology, 1987.

[58] R. Weller. "New Geometric Data Structures for Collision Detection". PhD thesis. Universität Bremen, 2012.

[59] R. Weller and G. Zachmann. "ProtoSphere: A GPU-assisted prototype guided sphere packing algorithm for arbitrary objects". In: *ACM SIGGRAPH ASIA 2010 Sketches, SA'10*. Jan. 2010.

Acknowledgments

This research has been funded by the Government of Spain through the grant BES-2013-062939 inside the FPI 2013 program.