ESCUELA POLITÉCNICA SUPERIOR

UNIVERSIDADE DA CORUÑA

Contact and HiL Interaction in Multibody Based Machinery Simulators

A thesis submitted for the degree of
Doctor Ingeniero Industrial

Alberto Luaces Fernández

Advisor: Daniel Dopico Dopico
Co–Advisor: Manuel González Castro

Ferrol, January 2013

Resumen

En la actualidad, el uso de simuladores está muy extendido en una vasta gama de disciplinas. Los simuladores permiten predecir y evaluar el comportamiento de sistemas y entornos complejos sin necesidad de disponer de una implementación material de los mismos.

Dependiendo del área de conocimiento en el que se emplee, el simulador es una herramienta que permite facilitar el proceso de diseño del sistema, o servir como instrumento de adiestramiento para los futuros usuarios del producto o del procedimiento.

Los simuladores son una valiosa herramienta en el proceso del diseño y fabricación de máquinas y mecanismos. La potencia actual de las herramientas de cálculo permite computar el comportamiento de un mecanismo ante las acciones de un usuario o el propio entorno de la simulación a una velocidad suficiente para garantizar la interactividad y la fidelidad con respecto al sistema real.

Motivación

Gracias a los prototipos computerizados, es posible detectar y corregir una gran parte de los fallos de diseño, evitando la costosa penalización que conlleva la construcción de un nuevo prototipo físico.

En los casos en los que el sistema mecánico sea controlado por un agente externo, como una persona o un sistema de control, el simulador puede cumplir las labores de sistema de entrenamiento. Las ventajas de los simuladores de entrenamiento con respecto al propio adiestramiento con el sistema mecánico real son las siguientes:

- La simulación conlleva un riesgo físico mucho menor. Si se adiestra a una persona en el manejo eficiente de una máquina, el entorno controlado de la simulación contribuye a que el aprendizaje se lleve a cabo virtualmente sin riesgos, en comparación con el manejo de la maquinaria por personal inexperto.

- Disminución de costes: el coste de los simuladores es generalmente mucho más reducido que el de las máquinas que emulan, tanto el coste de implementación como el de mantenimiento.

- Un simulador puede presentar escenarios críticos de actuación que serían difícilmente reproducibles en la realidad, dado el riesgo material y personal que podría conllevar la realización de un entrenamiento en esas circunstancias, o sencillamente su coste.

- Los simuladores computerizados facilitan el seguimiento y registro de las actividades de adiestramiento, facilitando el proceso de la monitorización de las actividades llevadas a cabo.

- La amortización de los equipos de simulación es más rápida que el coste equivalente de la maquinaria de entrenamiento. Una característica que contribuye a ello es el hecho de que la mayoría de los componentes del simulador de una máquina pueden ser compartidos por otros simuladores de otros tipos de maquinaria.

La dinámica multicuerpo contribuye a la implementación de simuladores de maquinaria cuyo entorno de funcionamiento puede ser demasiado complejo para ser emulado con modelos simplificados. En ese caso, la única solución es el empleo de modelos lo suficientemente detallados como para poder emular el proceso real. Por ejemplo, en el caso de la emulación de máquinas como excavadoras o grúas, que operan en condiciones muy variopintas, y donde existen múltiples factores de riesgo en cada una de ellas, es vital la determinación del estado instantáneo por parte del simulador. Las condiciones de carga, y las fuerzas de inercia que se derivan del movimiento de la máquina influyen en gran medida en su estabilidad, e incluso en la seguridad de la maniobra. Es necesario disponer de un modelo tridimensional fidedigno, que sea capaz de determinar el estado dinámico de la máquina en todo momento.

Existen simuladores de maquinaria basados en la dinámica multicuerpo, pero ciertamente es difícil encontrar documentación acerca de cómo adaptar e integrar dichos sistemas en un simulador. El proceso de la determinación del movimiento impone sus restricciones sobre el resto de los constituyentes del programa: la simulación numérica integra las ecuaciones del movimiento en unos pasos de tiempo determinados, así que, tanto las entradas a ese sistema como los datos extraídos del mismo, deben amoldarse a sus particularidades.

Metodología

Formulaciones multicuerpo

En primer lugar, se presenta una introducción a las diferentes opciones de formulación e integración del movimiento mediante sistemas multicuerpo. Se determina que la formulación de Lagrange Aumentado con proyecciones en *index-3* satisface los requerimientos de estabilidad y eficiencia que un simulador interactivo precisa. Esta formulación posibilita obtener la resolución del sistema diferencial-algebraico de las ecuaciones del movimiento para cada instante de tiempo. El esquema de integración empleado es el método *α-generalizado*, que disipa las altas frecuencias típicas en los sistemas multicuerpo, y alcanza la estabilidad incondicional y la precisión hasta el segundo orden, si se emplean los parámetros adecuados para el integrador.

Introducción de fuerzas de contacto

Por otra parte, la información de entrada al sistema multicuerpo debe haber sido transformada a las entidades con las que el sistema opera: fuerzas. La determinación de las actuaciones sobre el mecanismo son computadas con base en el contacto que pueda existir en cada momento entre las piezas de la máquina, o entre éstas y algún otro objeto presente en el entorno de la simulación. Generalmente, los modelos de contacto se desarrollan sin ningún tipo de restricción en cuanto al tiempo en el que una computadora pueda completar los cálculos. En un simulador interactivo, esta restricción existe y es, además, primordial para poder preservar la interactividad de éste. En el texto se describen modelos de contacto que permiten hallar las reacciones que se producirían entre pares de objetos que se encuentren en contacto, sin penalizar el rendimiento del simulador.

El modelo contemplado para la evaluación de las reacciones entre pares de objetos sigue las premisas del modelo original de Hertz. En concreto, se trata del modelo de Hunt-Crossley, que se vale de las velocidades de aproximación de los objetos, y del valor de la propia interpenetración entre ambos para establecer la acción correspondiente. El modelo contempla también la disipación de energía producida tras el contacto.

Para caracterizar los efectos de la fricción entre los objetos en contacto, se emplea un modelo basado en la ley de fricción de Coulomb, al cual se añade un término viscoso que representa la resistencia al deslizamiento debida a la fricción. Se incorpora también un modelo de fricción estática, *stiction*, que trata de representar la fricción a bajas velocidades de deslizamiento.

Ambos modelos se contrastan con los resultados experimentales de Bowden y Leben, verificándose satisfactoriamente su idoneidad para su empleo en simu-

ladores interactivos.

Se describe también un modelo simplificado de contacto con terreno deformable, que es de interés para maquinaria de movimiento de tierras: el modelo sirve para determinar las fuerzas derivadas de la actividad de excavación sobre zonas de terreno del entorno del simulador. Determinar las fuerzas derivadas de las labores de excavación es importante, puesto que pueden desestabilizar a la propia máquina. El modelo aproxima también el volumen de material extraído, de forma que posibilita la evaluación de la eficiencia en el trabajo del usuario del simulador.

Detección de objetos en contacto

Los modelos de contacto requieren, como ya se ha relatado, los parámetros que indiquen la disposición geométrica de los objetos en cada instante. Detectar qué objetos de la simulación están potencialmente en contacto, y cómo extraer los parámetros que los modelos de contacto necesitan, no es en absoluto una tarea trivial: el elevado número de objetos presentes en la simulación hace que comprobar todas las posibles combinaciones de elementos sea impracticable en un lapso de tiempo tan ajustado como del que se dispone en una simulación interactiva.

Esta fase es la denominada *fase de detección lejana*. Para solventar el problema, se emplean técnicas cuyo objeto es repartir el espacio del entorno de la simulación en celdas más pequeñas; éstas a su vez también son subdivididas, de tal manera que describen unas estructuras anidadas que facilitan un proceso de búsqueda más eficiente. Se describen las distribuciones de árbol octal (*Octree*), los árboles de división binaria (*BSP*, *Binary Space Partitioning Trees* en inglés) y los grafos acíclicos dirigidos (*DAG*, *Directed Acyclic Graph* en inglés). El uso de los *Octrees* y *BSP-Trees* se demuestra apropiado para la detección de contactos entre objetos móviles y objetos inmutables del escenario, mientras que los *DAG* son más adecuados para determinar las colisiones entre objetos móviles.

Una vez se determinen todos los pares de objetos en contacto, se debe determinar con precisión en qué puntos de la superficie se establece dicho contacto, y cuál es la dirección resultante de la reacción entre ambos. Esta fase es la llamada *fase cercana* o de detalle. Dependiendo del método exacto de definición de cada una de las superficies, se debe emplear un método específico de caracterización del contacto.

Los modelos más sencillos engloban los objetos dentro de volúmenes sencillos, como esferas o paralelepípedos. Su ventaja es que la detección detallada resulta muy sencilla, pero tienen el inconveniente de que suponen un alto coste en la precisión de los resultados, que se acentúa cuanto más complejas sean las superficies de los objetos.

Posteriormente se muestran modelos de colisión entre superficies cuya expresión analítica es algo más compleja, como cilindros y toroides, que sin embargo pueden ser de utilidad para la modelización de elementos específicos. En el texto se describe la posibilidad de emplearlos complementariamente para la simulación de contacto de eslabones de cadena, un elemento que está presente en muchos mecanismos.

Finalmente, se describe el establecimiento de los parámetros de colisión para superficies que se definan mediante mallas de triángulos. Se trata del método más genérico de los descritos, y que impone el menor esfuerzo de preproceso, dado que puede ser automatizado completamente, desde el modelado en un programa de diseño asistido hasta su incorporación al simulador. Dicha versatilidad tiene como contrapartida la dependencia de la precisión alcanzada de la resolución de la triangularización de la superficie.

Como colofón a la descripción de métodos de detección de contacto, se presenta un caso de implementación de deformación de un objeto descrito por una malla poligonal ante el contacto de otro descrito por una *envoltura convexa*: continuando con el ejemplo de la simulación de los procesos de movimiento de tierras, se presenta un modelo de que permite determinar la deformación de un terreno ante los procesos de excavación de una máquina. La superficie del cazo de la máquina excavadora se define a efectos de contacto por un poliedro convexo, lo que facilita las tareas de detección de contacto entre sus caras y la malla poligonal que constituye el terreno.

Sistemas “Human-” y “Hardware-in-The-Loop”

El desarrollo de un simulador conlleva la coordinación de los diferentes componentes, físicos o computacionales, de los que se compone. Los componentes físicos más críticos son los mandos, los dispositivos de representación gráfica y de audio. El uso —en la medida de lo posible— de elementos comerciales ya existentes permite abaratar el coste de implantación del simulador, a la par que añade la fiabilidad de contar con componentes con una cierta madurez en el mercado, lo cual minimiza el riesgo de fallo.

Existe un amplio catálogo de mandos y controles comerciales que emulan —e incluso en muchos casos se trata de las mismas piezas empleadas en la maquinaria real— la funcionalidad de los existentes en las máquinas reales. Gracias a ellos, la experiencia de los usuarios es más próxima a la realidad. Cuando ello no sea posible, existen también ciertos dispositivos que pueden llegar a ser comparables, al menos en funcionalidad. Por ejemplo, las pantallas táctiles son un buen sustituto para los tableros de mandos existentes en las cabinas de las máquinas de obra civil, puesto que pueden replicar su funcionalidad, y no se interponen en la capacidad de aprendizaje del sujeto.

Los medios de representación gráfica mediante proyección de imágenes son diversos, y el empleo de unos u otros depende en mayor o menor medida del tipo de simulador. Se describen dos tipos de proyección de imágenes, la plana y la esférica. La proyección plana es la más empleada, dado su coste asequible, sobre todo cuando se emplea la proyección tridimensional o estéreo. El modo *estéreo pasivo* emplea filtros polarizados para poder proyectar las imágenes correspondientes a los ojos izquierdo y derecho en la misma pantalla. El usuario lleva unas gafas con filtros, que permiten volver a separar las componentes individuales para cada ojo. Su desventaja es que asumen un observador cuya posición y orientación es fija en el espacio.

Las proyecciones en pantallas de casquete esférico permiten presentar una imagen cuya proyección es correcta, independientemente de la orientación de la visión del usuario. Su mayor inconveniente es el mayor coste de las pantallas y proyectores, dado que emplean lentes de gran angular para poder abarcar toda la pantalla. Por otra parte, la única técnica adecuada de representación estéreo es la denominada *estéreo activo*, que supone un mayor desembolso puesto que, tanto las tarjetas gráficas como las gafas necesarias con más complejas y más caras.

Las siguientes secciones tratan de describir cómo acomodar la lectura de los mandos y la representación gráfica de lo que sucede en el simulador a la simulación multicuerpo. En primer lugar, se expone el procedimiento de sincronización del algoritmo multicuerpo con el resto del programa: el bucle que computa el movimiento se ejecuta a una frecuencia mucho mayor que el resto de los eventos del simulador. Mientras que la integración del movimiento se hace en pasos de tiempo del orden del milisegundo, la representación gráfica del mismo puede ser uno o dos órdenes de magnitud más lenta. Se establece un algoritmo por el cual se prioriza el código multicuerpo sobre la representación, dado que es el componente más crítico del sistema. Asimismo, se muestra otro modelo de sincronización en el que es posible alterar la escala temporal real y la de la simulación, de tal manera que se pueda observar con mayor precisión algún evento de la simulación que fuese muy corto como para poder apreciarlo con detalle.

Existen ciertos detalles de la representación gráfica que pueden ser muy relevantes durante la operación del simulador. Uno de ellos es la capacidad de representar los espejos retrovisores de los que están dotados las cabinas de mando de las máquinas simuladas. De esta manera, el usuario del simulador goza de la misma perspectiva de la que se dispone cuando se usa la máquina real. Otro aspecto de mejora es el empleo de pantallas esféricas, puesto que permiten al usuario disponer del mismo campo de visión que en la realidad, independientemente de a dónde oriente su vista.

Ejemplo de implementación

El último capítulo expone un ejemplo real de simulador de maquinaria: una máquina retro-excavadora. El simulador constituye un sistema de entrenamiento completo, el cual no sólo reproduce el comportamiento de la máquina en un entorno de operación típico, sino que además se puede usar como herramienta de monitorización y evaluación de las capacidades adquiridas por el usuario durante su uso.

El simulador de retro-excavadora es un simulador que emplea las técnicas multicuerpo expuestas en los capítulos anteriores. Dichas técnicas permiten predecir el comportamiento de la máquina en un entorno típico de obra, en el cual —por la propia naturaleza de las actuaciones sobre el terreno— existen zanjas y desniveles, los cuales provocan las situaciones de riesgo que el adiestramiento sirve para que los usuarios aprendan a evitar.

En concreto, la caída o vuelco de la máquina es una de las causas más frecuentes de accidente. Es necesario simular estos procesos, ya que una de las habilidades que se pretende fortalecer es el conocimiento y empleo de los elementos estabilizadores de la máquina. En el caso del descenso de pendientes pronunciadas, es común entre el personal experimentado el empleo del propio cazo de la máquina a modo de *brazo* de apoyo con el fin de no perder la estabilidad. Es también frecuente el uso del cazo a la hora de salvar zanjas, puesto que éste permite elevar uno de los ejes de las ruedas de las máquinas y sortear el obstáculo.

Un modelo simplificado de excavadora podría, a lo sumo, detectar si la máquina está a punto de volcar en algunas situaciones, pero no podría determinar —dado que no dispone de la distribución de reacciones en cada pieza— si para una posición concreta de la máquina, el sistema hidráulico sería capaz de realizar la tarea encomendada.

Otro tipo de maniobras importantes en un simulador de este tipo es la emulación de las maniobras de excavación. El programa dispone de un modelo simplificado de terreno que puede ser empleado en tiempo real, y que determina las fuerzas de arrastre del cazo y el volumen excavado. Estos parámetros son de utilidad para poder evaluar la destreza del usuario del simulador.

El tipo de tareas que debe desempeñar el usuario para cada sesión está completamente abierto, dado que puede ser especificado en la configuración del programa. Mediante un sistema de *scripting*[1], es posible definir un amplio conjunto de tareas a realizar, evaluando en cada momento los parámetros del simulador con base en el criterio que se desee. De esta manera, se puede codificar el objetivo de la sesión de entrenamiento y las condiciones de éxito o fracaso mediante el lenguaje de *scripting*; este sistema permite desacoplar el código genérico del

[1]Programación mediante un lenguaje de alto nivel

simulador y la definición de las sesiones, proporcionando al mismo tiempo un entorno sencillo de programación.

Mediante un sistema de conexión en red, es posible comunicar a un programa de monitorización ejecutado por otro ordenador el estado instantáneo de uno o varios simuladores. Además, el inicio y la finalización de las sesiones también son controladas desde el puesto de monitorización, así como otros datos informativos, como la documentación del programa y de cada una de las tareas.

Conclusiones

El documento describe cómo el empleo de las técnicas multicuerpo es interesante para la simulación de maquinaria de un alto nivel de complejidad. El empleo de eficientes formulaciones, aptas para su empleo en programas interactivos, así como el uso de sistemas de detección de colisiones y los modelos de fuerza aplicados, permiten desarrollar simuladores con un alto grado de fidelidad al proceso emulado. Se ha hecho hincapié en el uso de componentes comerciales para la construcción de los simuladores, dada su disponibilidad inmediata y su reducido coste —al menos comparativamente— con respecto a sistemas específicamente diseñados para cada ocasión. El ejemplo propuesto del simulador de una retro-excavadora condensa la información contenida en los capítulos anteriores para mostrar un producto completo y versátil.

Abstract

Multibody simulators allow to predict and evaluate the motion of machines and mechanisms under the action of the user and the interaction with the simulated environment. Interactive simulators guided by a human or a piece of hardware must be efficient enough to compute the state of the system in real time. Therefore, employing fast and sufficiently accurate techniques is a must. In this work, generic tools for the implementation of this kind of simulators are provided.

Efficient multibody formulations are reviewed for implementing real-time simulators. The index-3 Augmented Lagrange formulation with projections of velocities and accelerations is selected, due to its efficiency and stability. The integration of the equations of motion follows the Generalized-α method, which provides high-frequency dissipation, and can be unconditionally stable and second-order accurate if suitable integrator parameters are chosen.

Contact modeling and detection is essential for computing the interaction among the mechanisms and the simulated environment. Normal and tangential contact force models are presented. For the normal contact, a Hertz-type Hunt-Crossley model is chosen. The tangential force model is based on Coulomb's law, and includes stiction and viscous friction effects. Both models were compared with the output of the Bowden-Leben stick-slip experiment. A real-time, simplified terrain model featuring digging forces for excavator simulators is also discussed.

Several techniques are shown for detecting colliding bodies at run-time. The collision detection process is divided into two stages. The first one is a broad range and coarse grained process, where potentially colliding pairs of objects are discovered. Spatial and hierarchical division techniques as Octrees, BSP-trees and Directed Acyclic Graphs are presented for this purpose. In the second stage, fine-detailed contact properties are computed from each pair of bodies. Several models are presented for testing object enclosing volumes or more complex surfaces discretized as triangular meshes.

State-of-the-art, Commercial Off The Shelf hardware devices are presented as the physical foundation of a simulator. Industrial-quality controllers, projection screens and audio devices are reviewed for this purpose. The implementation details for the use of those devices are also considered. Network communication procedures between the simulator and monitoring nodes are discussed, too.

Finally, a particular implementation of all the techniques described in previous chapters is presented in the form of an interactive excavator simulator, which features all the degrees of freedom of the machine, and is able to perform earthmoving operations in a realistic environment. Monitoring capabilities are also available, and any training session can be defined by user scripts.

The techniques described in this document constitute a generic and efficient compendium of algorithms that are well-fitted for medium or low-end computational systems, as desktop or even laptop computers.

Contents

Chapter 1

Introduction

Simulation is a broad field of development, whose aim is to replicate any kind or process for its study. Its main advantage is to be able to analyze a process without having to originate it materially. Proof of this is the existence of simulator development in very heterogeneous knowledge areas, see for example [66, 85, 76]. For mechanical engineering, it is also a topic of great interest. The design of a machine or mechanism can be validated through the use of prototype implementations that are used to verify the initial premises of the project. Simulation methods aid in that task, by reducing the need for building prototypes at a minimum level, since most of the verification stage can be developed by simulation procedures.

Human-driven machine manufacturers and sellers can also benefit from simulations: simulators can serve as well as a training tool for future operators, lowering costs and risks, and being considerably less expensive than a real training session.

There is a high demand for vehicle training simulators. Aerospace and airplane simulators constituted the first learning platforms that allowed the operators to get acquainted with the complexity of the controls of the vehicle without risking their lives or the real machine [92]. Nowadays they are an indispensable part of the product development cycle, and have a parallel development with respect to the vehicle itself [11]. Military simulators make it possible to train in the use of helicopters, tanks, airplanes, while following tactical procedures.

As the prices for computing platforms decay while their computing power rises, training simulators are becoming more popular, even destined to replicate the behavior of less expensive machinery. Buses, cars, excavators, trucks... can be simulated and save hours of real machine operation. The cost of those simulators is dependent on the fidelity of the system, but tends to be affordable for training schools.

Notwithstanding, most of those simulators need not to be physically accurate: a high degree of computing power is saved by using reduced or simplified models, which offer a similar experience to the real behavior of the machine. For scenarios whose operation environment is tightly bounded, such as simulators where inputs are always constrained to known values, that simplification is highly beneficial. For example, an automobile driving simulator aimed for students that want to obtain their driving license can disregard the complexity of a real vehicle model, and use a simplified one instead. The environment does not allow the user to perform any action for which the difference in the behavior of both models would be noticeable.

There are, however, processes that can not be replicated at an acceptable degree of realism using simplified models. Those simulators have to respond to a high number of different types of events, and the possible configurations of the machine can be too complex to be represented by a simple model. An off-road vehicle simulator must represent the whole suspension system in order to be able to replicate the position and the available traction for the vehicle at any instant of time. The operating simulator for a machine moving loads (such as an excavator or a crane), must take into account the inertial forces that show up during the movement, and that are determinant for the safe operation of the machine. The complex configuration of these machines makes sometimes not practical, or even impossible, to develop simple models able to attain a certain level of accuracy.

For machine simulators, it is therefore necessary to make use of tools that can aid to solve mechanical problems in a generic way. Multibody dynamics is a recent albeit mature discipline that is able to compute the motion of a mechanical system considering the interconnected bodies or parts that it is made of. Multibody formulations can aid to define the mechanical system part by part in a computer-friendly way, and numerically integrate the equations of motion, thus obtaining the behavior of the machine as an output. Currently, there exist several successful multibody-based simulators; see for example [42], [93], or [73].

The input of the multibody system is constituted by the forces exerted over the mechanism. Those forces are given by the interaction of the mechanism with its environment, or from the simulation of on-board devices of the machine, e.g. an engine. A machine simulator is meant to depict the interaction of a device within an environment, thus rendering contact events with the scenery or other objects as the most relevant phenomenon to characterize.

The results of the training simulator have to be presented to the user at a rate that allows interaction to exist between the human and the machine. Display systems must depict the environment and the simulated machine from the point of view of the user, mimicking real operative sessions. The user has to respond to those stimulus and act over the controllers that send back operation commands to the simulator, in order to guide the machine.

1.1. Motivation

Simulators are an excellent tool for training in the use or designing a mechanical system. Their main advantages can be listed as:

- Diminish physical risks of damage for the operators of the machines, being the simulator in a controlled environment.
- Reduce the costs of training by reducing the rental costs of real machinery. Users can get fully used to the simulated machine before trying the real machine for the first time.
- Simulated operation environments can be vastly diverse, and can represent situations that are very difficult or impossible to train in real life.
- Simulators ease the task of monitoring the learning process, and logging it for later review.
- Hardware expenses can be recovered faster since the same equipment can be shared among different simulators. Learning sessions with real machines requires each different machine to be acquired separately.

Simulators can benefit from the use of multibody techniques in order to be able to reproduce the motion of complex machines. Risky maneuvers can be simulated, e.g. rollovers. Forces acting over each part of the machine can be computed, and it can be established if a certain maneuver could damage the machine. It can be established the performance of the motors given the current load of the machine.

Unfortunately, the integration of a multibody system into a simulator infrastructure is not a trivial task. Multibody systems require precise information from the instantaneous state of the simulator in order to predict the position of the bodies in the scene. Fast an accurate contact models must be used in order to simulate the interaction between bodies. User's actions have to be read by the simulator, and transmitted to the multibody model to guide the machine. The graphics system that renders the three-dimensional scene from the point of view of the user must be synchronized with the multibody system, since their timings can differ in orders of magnitude.

There are scarce resources that cover the topic of interactive multibody simulators while providing general purpose techniques that can be adapted to simulate virtually any kind of mechanism.

1.2. Objectives

The objective of this work is to establish a reference for implementing multibody-based simulators, providing some guidelines for common development stages:

- Establish a flexible and versatile multibody methodology as the basis for simplified simulator development.
- Research physical contact models that can be used to solve contact problems in the simulation, while at the same time being sufficiently efficient to be run at interactive rates.
- Research the use of adequate collision detection techniques in order to be able to incorporate contact and impact events into the simulation. Develop additional models that can be needed in special situations (earth-moving operations).
- Determining the necessary hardware requirements for a complete simulation experience. Prefer, if possible, using *Commercial Off-The-Shelf* (COTS) devices over specific and usually more expensive solutions.

1.3. Overview

This document is divided into the following chapters:

Chapter 2 contains a brief introduction to the multibody formalism, including the description of several families of coordinate types, equation formulations and numerical integrators.

Chapter 3 presents some suitable contact models that can be used in a satisfactory way as a building part of interactive simulators, allowing to compute reaction and friction forces.

Chapter 4 describes algorithms for detecting pairs of simulated objects in collision, and how to craft the required parameters that are needed for computing the contact forces.

Chapter 5 includes a discussion about different types of COTS hardware that can be used to implement a simulator, and techniques for creating a realistic session environment for the user.

Chapter 6 is an example of the application of the techniques described in this document to real, complex problems. As an example, an excavator machine training simulator is presented. The description covers all the techniques needed to develop the simulator, and the specific adaptations that had to be performed for this specific case. As an example, the simulator features a logging system which allows an instructor to follow a training session remotely, even taking notes about student's performance and grading it. The interface for the instructor and the network communication mechanism between the computers is described in detail.

Chapter 2

Multibody Dynamics Formulation

2.1. Multibody dynamics

Modern engineering challenges involve the enhancement of traditional methodologies through the adoption of currently available computing tools. Traditional mathematical models were required to provide human-computable solutions, either by means of analytic expression models or simplified and specialized algorithms. In the motion analysis field, traditional methods offer analytic solutions for the simplest cases, or specific analysis of the system at a concrete configuration and time instant. Engineer expertise is determinant to be able to identify the most suitable method and the most critical situations for the system. However, there exist cases where heterogeneous phenomena problems cannot be simplified, or it is very difficult to systematically adapt the method for any different kind of problem that it could be desired to solve.

Machinery simulators are an example of those complex systems, where a high number of external inputs and multiple configurations for the machine's state have to be considered.

Classical Mechanics benefited from the elegant and powerful analytic mechanics approach, aiming to solve the equations of motion at a higher level just by finding the equations for the energy of the system, expressed in the terms of independent coordinates called *generalized coordinates*. The Lagrangian method avoids the need for computing all the element-wise dynamic equilibrium of forces, therefore reducing the number of equations. Nevertheless, the resulting differential equations are, in general, very difficult to solve, and there exists the possibility that they can not be calculated analytically. In addition, for some particular generalized coordinates selections, the resulting kinematic equations can become very complex expressions. This is the case for closed-loop mechanisms, where the kinematic expressions of each body are not trivially or —at least— easily expressed in terms of the generalized coordinates.

Multibody dynamics is an inherently computerized discipline which solves those aforementioned generic problems: the definition of a mechanical system by sets of coordinates, independent or not, and the integration of the resultant motion equations over time. Multibody systems are designed to be solved by computer aided means, therefore favoring generic, simple algorithms over brief although complex methods. Many multibody software packages do not even require that the analyst is able to define the coordinates chosen for the system modeling. The definition of the system can be done just by defining each of its parts, and the kind of joints that make up the mechanism. Nevertheless, when performance is a critical factor, specially in real-time scenarios, a careful choice of the coordinates and formulations to be used can still make a significant difference.

In fact, multibody techniques are diverse: there exist many different coordinate, formulation, and integrator families, see [69], [59] or [97]. Those families can be combined in many ways when aiming for the best performance for a particular system. Multibody techniques can also be integrated with other models describing other types of phenomena as FEM or CFD. This integration can happen at the system level, by building an equation system containing all the equations for the different phenomena, or by solving each system independently, and then exchanging information at every integration time step (co-simulation), see [81] and [50].

2.1.1. Multibody coordinates

There exist many families of coordinates commonly employed in multibody systems. The traits that characterize those families are its ease of use at modeling stage, the number of equations that each family generates and the computation cost that imposes each of those equations due to the choosing of a particular coordinate family.

The modeling coordinates easiness allows to have a short and straightforward modeling phase, since the coordinates fit better into the system. Should this not be the case, the multibody user would have to spend more time and perform many more operations in order to define the system.

Every family of coordinates imposes a certain degree of redundancy; as a general rule, the less the redundancy the better, since this imposes more constraint equations. On the other hand, less redundancy usually implies less flexibility.

The computational cost for each coordinate family can be measured on the added complexity imposed to the final motion equations. Some coordinate families can describe in a simpler way certain types of motion, while certain others —for example three-dimensional rotations— can be complex to manipulate.

As a example of several coordinate families, the following are presented:

- Reference point coordinates [59]: this type of coordinates could be though as the most simple ones, since they follow the traditional free body scheme. They define the position and the orientation of each body through the three Cartesian coordinates of one of its points and a set of rotation parameters. The biggest disadvantage of this coordinates lies in the need for a rotation parametrization. A free body in the space has six degrees of freedom, corresponding to three for the position and three for the rotation. Unfortunately, all three-parameter rotation descriptions are subject to singularities for certain orientations. Many singularity-free parametrizations have been developed, but at the cost of having to use at least one extra parameter, and the corresponding normalization constraint. Axis-angle parametrization and quaternions are examples of those rotation representations. Furthermore, the computation with those coordinate sets is more complex: for some operations, a consistency check or normalization is required, and composition operations are not trivial.

- Relative or joint coordinates [30]: the aim of this family of coordinates is to use the least number of coordinates for the whole system. The motion of a body is defined relative to the body or frame that it is attached to, and this allows to only take account of the relative motion that the joint permits. For example, a body attached by a revolute joint to its neighbor could be modeled with only one coordinate representing the relative angle between them. The disadvantages show up when dealing with mechanisms that have closed-loop topology, since the kinematics become very complex and they have to be split for obtaining open chains that can be defined as a tree. In addition, the calculation of the dynamical terms is complex. This overhead usually makes these kind of coordinates to be useful in large systems with a low number of degrees of freedom per body, or with an open chain or tree-like topology, since they lead to very compact systems.

- Natural coordinates [69]: those coordinates can be regarded as very versatile at the mechanism definition stage. Natural coordinates are used as sets of points and unit vectors. The philosophy of these coordinates is to define the mechanism locating points and vectors in the joints of the system, instead of trying to model each body independently. Thus, the coordinates placed in the joints, will be shared by two or more bodies simultaneously. Defining three or more entities — any combination of points or vectors— for each body completes its parametrization, avoiding to have to deal with rotations directly: the body frame is completely defined by the points or vectors. Despite the easiness of the modeling process, the points and vectors form a redundant set of variables, which must be constrained to enforce the rigid body behavior of the bodies. Two kind of rigid body constraints appear: a first set related to unitary vectors or constant distances between points; a second set related to constant orientation of points and vectors. Moreover some additional constraints are usually needed for some kind of joints that cannot be imposed by sharing points or vectors. The resulting constraints are usually linear or quadratic at most, and they do not contain trigonometric functions. Systems modeled with natural coordinates usually lead to big, sparse systems of equations. If a sparse solver is not available and the system is moderately big, this can incur into a penalization in performance.

Many software packages assist the user in the task of designing mechanisms through graphical user interfaces or configuration files. It is very difficult to develop a system like this using natural coordinates because it does not exist one unique solution: a mechanism could be programmed in several different ways depending on the chosen set of coordinates. Human hindsight is usually required to find good, sensible sets of coordinates that can provide the best possible performance.

- Mixed coordinates: this is a superset of the natural coordinates. It can be convenient to have some scalar variables as angles or distances for simplicity purposes: many force models or controller algorithms require those measurements as input parameters. Having those coordinates allows the user to inspect some displacements or rotations, or provide their values to other systems without having to preform extra computations. On the other hand, the constraints that relate the coordinates to the rest of the mechanism are no longer linear, rendering the Jacobian of the derivatives more complex.

2.1.2. Multibody formulations

Multibody formulation is referred here, basically, as the final form of the motion equations used to solve the dynamics of multibody systems. Formulations comprise aspects like the choice of coordinates, the basic mechanical principles used to derive the motion equations and the way to enforce the constraint equations. They also influence the manner in which other parameters derived from the motion such as reaction forces can be obtained. In the spirit of the multibody discipline, sometimes a formulation presenting a low level of performance can be employed if it facilitates the computation of some desired quantities such as reaction forces. Other formulations can have a greater performance and stability by requiring the computation of additional terms. The analyst will have to decide if it is worth this additional complexity. In the following, some formulations are cited:

- Direct solving of the motion equations: the equations of the model result in an index-3 differential-algebraic system of equations (DAE) that it is not is usually directly solved because of the numerical difficulties involved, see [14, 1]. There exist some solver packages that are able to integrate those systems, first converting them into an index-1 system. An example is the *Differential-Algebraic System Solver* DASSL by [87] and described by [1].

- Stabilized Lagrange: transforms the DAE into a index-1 DAE system, which is solvable by *Ordinary Differential Equations* (ODE) methods. The transformation is accomplished by deriving two times the constraint equations. As a side effect, this leads to unstable systems as the derivatives of the constraints are enforced, instead of the constraints themselves. Therefore, the obtained solutions can violate slightly the constraints every time step following a quadratic function in time, until the system ceases converging.

- Baumgarte stabilization: to overcome this issue, [4] proposed a stabilization of the constraint derivatives that leads to more robust systems. However, depending on the particular application, the energy loss caused by the dissipation is inadmissible.

- Projection Matrix $\mathbf{R}$ [96, 30]: the difficulty of solving the DAE can be avoided if somehow the system is defined in terms of independent coordinates only. For each time step, a linear transformation between dependent and a set of independent coordinate velocities can be found, the matrix $\mathbf{R}$. This allows to define the equations of motion only in terms of the independent coordinates and its derivatives. The resulting system is an ODE. The two big disadvantages of this formulation is the overhead of the $\mathbf{R}$ matrix computation, and its instability when approaching to singular positions where the number of degrees of freedom increases. On the other hand, having an ODE system avoids all the difficulties related to the integration of DAE systems. In addition, this is a specially well-suited formulation for computing motor efforts since the contribution of or over each coordinate is unequivocal.

- Penalty formulations [6]: this formulation follows the spirit of the penalty constrained optimization methods. Those methods introduce *penalty* forces proportional to the degree of violation of the constraints. This is accomplished by choosing very high numerical values and scaling the constraint violation times those values. Penalty terms represent the stiffness of the springs opposing to the motion in any direction perpendicular to the constraint manifolds. Furthermore, the system provided by this formulation is smaller than the stabilized Lagrange method. Another advantage is that the formulation can deal with redundant constraints and singularities. Its disadvantage lies on the poor numerical conditioning that the penalty factor imposes, specially for high values of the penalty factor. On the other side, choosing too low penalty factors would result in inaccurate solutions.

- Augmented Lagrange [5]: this formulation constitutes an improvement over the penalty formulation since it combines the latter with the Lagrange multipliers method. Thus, smaller penalty factors can be used, enforcing better numerical conditioning, while at the same time reaching to the exact —as far as the numerical implementation allows— solution. In this occasion, an extra iteration loop must be performed in order to update the Lagrange multipliers, but since most of the terms are constant in the iteration loop, the computational overhead is small. In addition, if at a later integration stage it is also needed to perform an iteration loop, those two loops can be performed simultaneously with a reduced additional computational cost.

Along the projects described in this document, an *index-3 Augmented Lagrangian formulation with projection of velocities and accelerations* is used. This technique allows to solve in an efficient and compact manner the motion problem. This formulation has been extensively used and tested by [7], [23] and [30].

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 [69]:

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

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

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

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

This is a Differential-Algebraic Equation system (DAE) that it is not usually solved directly. Non-linear DAEs with a high index lead to very complex solving algorithms, as described in [14] and [57]. This complexity entails further disadvantages as equation instability problems, and the requirement of specific integration methods.

For multibody dynamics, this complexity can be avoided 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 those 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 an Ordinary Differential Equation system (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

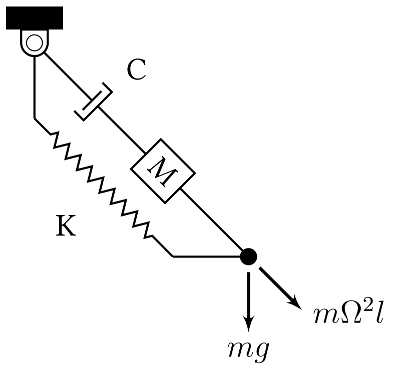


Figure 2.1: Physical meaning of penalty factors.

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.

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

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. Therefore, they are set to $\omega = 10$ and $\xi = 1$ for critical dissipation.

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

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

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

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 on the projection of $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$ over the manifolds represented by $\dot{\boldsymbol{\Phi}}$ and $\ddot{\boldsymbol{\Phi}}$.

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:

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

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

As mentioned, Lagrange multipliers $\boldsymbol{\lambda}_{i}^{*}$ are computed according to the current state of the system, and it 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 for 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 [83], HHT [64] and Generalized-α [19] are the most common ones. The expression for these implicit integrators can be used to compute positions from the coordinate values and its derivatives of the previous time step. For example, the Newmark integrator family,

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

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

$\ddot{\mathbf{q}}_{n+1}$ can be isolated as

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

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

Introducing them into (2.10), a non-linear equation system is presented:

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

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

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

$$\mathbf{q}_{i+1} = \mathbf{q}_i + \Delta\mathbf{q}_{i+1} \quad (2.18)$$

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

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

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 within the rest of the matrix values. $\mathbf{\Phi}^{\mathrm{T}}_{\mathbf{qq}}$ 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 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

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

being

$$\mathbf{K} = -\frac{\partial \mathbf{Q}}{\partial \mathbf{q}} \quad (2.21)$$

$$\mathbf{C} = -\frac{\partial \mathbf{Q}}{\partial \dot{\mathbf{q}}} \quad (2.22)$$

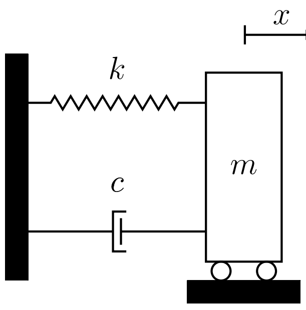


Figure 2.2: Harmonic oscillator

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 $\frac{\partial F}{\partial x} = k$. Constant c behaves in the same manner with respect to the displacement time derivative, $\frac{\partial F}{\partial \dot{x}} = c$. 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). A factor of $h^2/4$ is used in order to avoid having to scale the mass matrix:

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

As noted in [23], the approximate tangent matrix (2.20) can be ill-conditioned 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.

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 [7], which described the projection of velocities and accelerations over the mass matrix. The original formulation is

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

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 a 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 equation system:

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

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 [84].

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. [24] 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:

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

Substituting the weight matrix P with its own value, the final expression is

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

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, so a large number of computer cycles is saved by means of this technique. 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 $\ddot{q}$ are projected over a matrix and required to fulfill the constraints. For the mass matrix-orthogonal projection:

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

$$\text{subject to } \ddot{\mathbf{\Phi}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) = 0$$

And for the corresponding Augmented Lagrangian method:

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

Here ν are the Lagrange multipliers for the problem of the acceleration projection. The expression resulting from differentiating this equation in order to find its minimum:

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

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

Again, using an iterative method which updates the multipliers in the form $\nu_{i+1} = \nu_i + \alpha\dot{\mathbf{\Phi}}_{i+1}$. However, in the same way as the velocities projection, a simpler problem can be solved using only a penalty method. Therefore, a linear equation system is obtained:

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

The *tangent matrix* can be used as well in the acceleration's case by redefining the problem in terms of the 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:

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

2.3.2. Generalized-α integrators

So far, the Newmark family of integrators was presented in order to solve the equations of motion by the index-3 Augmented Lagrangian method. With this integrator family, a set of parameters can be chosen in order to reach an unconditionally stable integration with a precision up to the second degree for linear systems. This is the case for the *trapezoidal rule*, a specific integrator obtained out of the Newmark family when $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$. Unfortunately, as mentioned the *trapezoidal rule* is not unconditionally stable when applied to non linear problems, or when the problem includes constraints. When any of those scenarios is met, different values for γ and β have to be chosen in order to ensure the stability of the method, at the cost of numerical dissipation and the loss of second order accuracy. In some cases, those disadvantages are not admissible, and more advanced integrators are required.

Integrator families as the *Hilbert-Hughes-Taylor* (HHT) algorithm or the *Generalized-α* were developed in order to reach second order accuracy and still be able to dissipate stiff or high frequencies introduced by the constraints. The HHT method is a generalization of the Newmark family, and the *Generalized-α* is a generalization itself of the HHT method. Thus, the HHT method will only be briefly presented here as a reference step for the *Generalized-α* method description.

The HHT algorithm uses the same equations as the Newmark family, but if the latter defined the motion equation as

$$\mathbf{M}_{n+1}\ddot{\mathbf{q}}_{n+1} - \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}})_{n+1} = \mathbf{0}, \tag{2.42}$$

the HHT method transforms it into

$$\mathbf{M}_{n+1}\ddot{\mathbf{q}}_{n+1} - (1-\delta_f)\mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}})_{n+1} - \delta_f\mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}})_{n} = \mathbf{0} \tag{2.43}$$

where δ_f is a new, specific parameter of the method. The three parameters, β and γ from Newmark, and δ_f from HHT, are related by the following identities:

$$\gamma = \frac{1+2\delta_f}{2} \tag{2.44}$$

$$\beta = \frac{(1+\delta_f)^2}{4} \tag{2.45}$$

$$\delta_f \in \left[0, \frac{1}{3}\right] \tag{2.46}$$

As seen in (2.43), the HHT method interpolates the external force vector $\boldsymbol{Q}(q, \dot{q})$ between the previous and the next time step. The *Generalized-α* method further modifies the motion equations by interpolating not only the forces depending on positions and velocities, but also those depending on the accelerations:

$$(1-\delta_m)\mathbf{M}_{n+1}\ddot{\mathbf{q}}_{n+1} + \delta_m\mathbf{M}_n\ddot{\mathbf{q}}_n - (1-\delta_f)\mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}})_{n+1} - \delta_f\mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}})_{n} = \mathbf{0} \tag{2.47}$$

being δ_m the special parameter carried by this method. In a similar way to the HHT algorithm, the four parameters for the *Generalized-α* method can be rewritten in terms of only one parameter, the *spectral radius* ρ_∞:

$$
\begin{aligned}
\delta_m &= \frac{2\rho_\infty - 1}{\rho_\infty + 1} \\
\delta_f &= \frac{\rho_\infty}{\rho_\infty + 1} \\
\gamma &= \frac{1}{2} - \delta_m + \delta_f \\
\beta &= \frac{1}{4}(1 + \delta_f - \delta_m)^2 \\
\text{with } & \rho_\infty \in (0, 1]
\end{aligned}
$$

Setting $\rho_\infty = 1$ provides the maximum energy dissipation possible, and values towards zero dissipates the least. Account must be taken into the fact that, for constrained systems, ρ_∞ can not be set to zero, since the system would become unstable [56]. When applied to the index-3 Augmented Lagrangian method, the motion equations present the form:

$$\mathbf{M}\ddot{\mathbf{q}}_{\delta m} + \left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\lambda}^* + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\mathbf{\Phi}\right]_{\delta f} = \mathbf{Q}_{\delta f} \tag{2.48}$$

$$\ddot{\mathbf{q}}_{\delta m} = (1 - \delta_m)\,\ddot{\mathbf{q}}_{n+1} + \delta_m\ddot{\mathbf{q}}_n \tag{2.49}$$

$$\mathbf{Q}_{\delta f} = (1 - \delta_f)\,\mathbf{Q}_{n+1} + \delta_f\mathbf{Q}_n \tag{2.50}$$

$$[\ldots]_{\delta f} = (1 - \delta_f)\,[\ldots]_{n+1} + \delta_f[\ldots]_n \tag{2.51}$$

$$\boldsymbol{\lambda}_{i+1}^* = \boldsymbol{\lambda}_i^* + \alpha\mathbf{\Phi}_{i+1}, \; i = 0, 1, 2, \ldots \tag{2.52}$$

In equation (2.48), the inertia, constraints and external forces are weighted between the steps n and $n+1$ following the expressions (2.49), (2.50), (2.51), δ_f and δ_m are parameters of the Generalized-α method.

The integrator difference equations are the Newmark expressions [83].

$$\dot{\mathbf{q}}_{n+1} = \frac{\gamma}{\beta h}\mathbf{q}_{n+1} + \hat{\dot{\mathbf{q}}}_n \tag{2.53}$$

$$\ddot{\mathbf{q}}_{n+1} = \frac{1}{\beta h^2}\mathbf{q}_{n+1} + \hat{\ddot{\mathbf{q}}}_n \tag{2.54}$$

$$\hat{\dot{\mathbf{q}}}_n = -\left[\frac{\gamma}{\beta h}\mathbf{q}_n + \left(\frac{\gamma}{\beta} - 1\right)\dot{\mathbf{q}}_n + \left(\frac{\gamma}{2\beta} - 1\right)h\ddot{\mathbf{q}}_n\right] \tag{2.55}$$

$$\hat{\ddot{\mathbf{q}}}_n = -\left[\frac{1}{\beta h^2}\mathbf{q}_n + \frac{1}{\beta h}\dot{\mathbf{q}}_n + \left(\frac{1}{2\beta} - 1\right)\ddot{\mathbf{q}}_n\right] \tag{2.56}$$

where h is the time-step and β, γ are integrator parameters.

Replacing (2.53) and (2.54) in (2.48).

$$\begin{aligned}\mathbf{M}\left[(1-\delta_m)\left(\frac{1}{\beta h^2}\mathbf{q}_{n+1}+\hat{\ddot{\mathbf{q}}}_n\right)+\delta_m\ddot{\mathbf{q}}_n\right]+\\ (1-\delta_f)\left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\left(\alpha\mathbf{\Phi}+\boldsymbol{\lambda}^*\right)-\mathbf{Q}\right]_{n+1}+\delta_f\left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\left(\alpha\mathbf{\Phi}+\boldsymbol{\lambda}^*\right)-\mathbf{Q}\right]_n=\mathbf{0}\end{aligned} \tag{2.57}$$

In the last equation, $\mathbf{q}_n$, $\dot{\mathbf{q}}_n$ and $\ddot{\mathbf{q}}_n$ are known quantities coming from the previous time-step, n, therefore $\hat{\dot{\mathbf{q}}}_n$ and $\hat{\ddot{\mathbf{q}}}_n$ are also known, and $\mathbf{q}_{n+1}$ are the only unknowns to solve in the present time-step $n+1$. Thus (2.57) is a nonlinear system of equations that can be solved using the Newton-Raphson iteration.

$$\left[\frac{\partial f\left(\mathbf{q}\right)}{\partial\mathbf{q}}\right]_i\Delta\mathbf{q}_{i+1}=-\left[f\left(\mathbf{q}\right)\right]_i \tag{2.58}$$

$$\boldsymbol{\lambda}^*_{i+1}=\boldsymbol{\lambda}^*_i+\alpha\mathbf{\Phi}_{i+1} \tag{2.59}$$

being,

$$\begin{aligned}f(\mathbf{q})=\beta h^2\left\{\mathbf{M}\left[(1-\delta_m)\ddot{\mathbf{q}}_{n+1}+\delta_m\ddot{\mathbf{q}}_n\right]+\right.\\ \left.(1-\delta_f)\left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\left(\alpha\mathbf{\Phi}+\boldsymbol{\lambda}^*\right)-\mathbf{Q}\right]_{n+1}+\delta_f\left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\left(\alpha\mathbf{\Phi}+\boldsymbol{\lambda}^*\right)-\mathbf{Q}\right]_n\right\}\end{aligned} \tag{2.60}$$

$$\left[\frac{\partial f(\mathbf{q})}{\partial\mathbf{q}}\right]\cong(1-\delta_m)\mathbf{M}+(1-\delta_f)\gamma h\mathbf{C}_{n+1}+(1-\delta_f)\beta h^2\left(\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\mathbf{\Phi}_{\mathbf{q}}+\mathbf{K}\right)_{n+1} \tag{2.61}$$

where,

$$\mathbf{K}=-\left(\frac{\partial\mathbf{Q}}{\partial\mathbf{q}}\right) \tag{2.62}$$

$$\mathbf{C}=-\left(\frac{\partial\mathbf{Q}}{\partial\dot{\mathbf{q}}}\right) \tag{2.63}$$

For efficiency reasons, advantage is taken from the Newton-Raphson iteration (2.58) to update the Lagrange multipliers (2.59), together in the same iteration. Again, to improve the numerical conditioning, the equation (2.57) was scaled by a $\frac{h^2}{4}$ factor.

Projections of velocities and accelerations

As with the the Newmark integrator case, the fields of dependent velocities and accelerations coming from the integration stage of §2.2.2, have also to be forced to fulfill the constraint equations derivatives $\dot{\mathbf{\Phi}} = \mathbf{0}$, $\ddot{\mathbf{\Phi}} = \mathbf{0}$. The projections presented here are a generalization of the original ones presented in [7] and of the modified projections, presented in [24].

The problem to solve for the velocities is the following.

$$\min V = \frac{1}{2}\left(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*\right)^{\mathrm{T}} \mathbf{P}\left(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*\right) \tag{2.64}$$

$$\text{subject to } c\dot{\mathbf{\Phi}}\left(\mathbf{q}, \dot{\mathbf{q}}, t\right) = 0 \tag{2.65}$$

where $\dot{\mathbf{q}}^*$ are the velocities that do not fulfill the constraint equations, $\dot{\mathbf{q}}$ are the projected velocities resulting from the projection, $\mathbf{P}$ is the weight matrix (or projection matrix) and c is a constant for the weight of the constraints.

An option to deal with the constrained problem, is to use an Augmented Lagrangian formulation to transform the constrained minimization problem (2.64) into another equivalent unconstrained system.

$$\min_{\dot{\mathbf{q}}} V^* = \frac{1}{2}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{P}\left(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*\right) + \frac{1}{2}c\dot{\mathbf{\Phi}}^{\mathrm{T}}\boldsymbol{\alpha}\dot{\mathbf{\Phi}} + \dot{\mathbf{\Phi}}^{\mathrm{T}}\boldsymbol{\sigma} \tag{2.66}$$

where $\boldsymbol{\sigma}$ is the vector of Lagrange multipliers of the minimization problem. The necessary condition to obtain the minimum is the following.

$$\begin{aligned}\frac{\partial V^*}{\partial \dot{\mathbf{q}}} = \mathbf{P}\left(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*\right) + c\dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}}^{\mathrm{T}}\boldsymbol{\alpha}\dot{\mathbf{\Phi}} + \dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}}^{\mathrm{T}}\boldsymbol{\sigma} = \mathbf{P}\left(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*\right) + c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\dot{\mathbf{\Phi}} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\sigma} = \\ \mathbf{P}\left(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*\right) + c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\left(\mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} + \mathbf{\Phi}_t\right) + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\sigma} = \mathbf{0}\end{aligned} \tag{2.67}$$

In (2.67), the following kinematic relations were used.

$$\dot{\mathbf{\Phi}} = \mathbf{\Phi}_{\mathbf{q}}\dot{\mathbf{q}} + \mathbf{\Phi}_t \tag{2.68}$$

$$\dot{\mathbf{\Phi}}_{\dot{\mathbf{q}}} = \frac{\partial \dot{\mathbf{\Phi}}}{\partial \dot{\mathbf{q}}} = \mathbf{\Phi}_{\mathbf{q}} \tag{2.69}$$

$$\tag{2.70}$$

where $\mathbf{\Phi}_t = \dfrac{\partial \mathbf{\Phi}}{\partial t}$.

Expression (2.67) is a nonlinear system of equations that can be solved using the fixed-point iteration.

$$\left(\mathbf{P} + c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\mathbf{\Phi}_{\mathbf{q}}\right)\dot{\mathbf{q}}_{i+1} = \mathbf{P}\dot{\mathbf{q}}^{*} - c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\mathbf{\Phi}_{t} - \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\sigma}_{i+1} \tag{2.71}$$

$$\boldsymbol{\sigma}_{i+1} = \boldsymbol{\sigma}_{i} + c\boldsymbol{\alpha}\dot{\mathbf{\Phi}} \tag{2.72}$$

The problem to solve for the accelerations is the following.

$$\min V = \frac{1}{2}\left(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*}\right)^{\mathrm{T}}\mathbf{P}\left(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*}\right) \tag{2.73}$$

$$\text{subject to } c\,\ddot{\mathbf{\Phi}}\left(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t\right) = 0 \tag{2.74}$$

where $\ddot{\mathbf{q}}^{*}$ are the accelerations that do not fulfill the constraint equations, $\ddot{\mathbf{q}}$ are the projected accelerations resulting from the projection, P is the weight matrix (or projection matrix) and c is a constant for the weight of the constraints.

Proceeding like in the projections of velocities.

$$\min_{\ddot{\mathbf{q}}} V^{*} = \frac{1}{2}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*})^{\mathrm{T}}\mathbf{P}\left(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*}\right) + \frac{1}{2}c\ddot{\mathbf{\Phi}}^{\mathrm{T}}\boldsymbol{\alpha}\ddot{\mathbf{\Phi}} + \ddot{\mathbf{\Phi}}^{\mathrm{T}}\boldsymbol{\kappa} \tag{2.75}$$

where $\boldsymbol{\kappa}$ is the vector of Lagrange multipliers of the minimization problem. The necessary condition to obtain the minimum is the following.

$$\frac{\partial V^{*}}{\partial \ddot{\mathbf{q}}} = \mathbf{P}\left(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*}\right) + c\ddot{\mathbf{\Phi}}_{\ddot{\mathbf{q}}}^{\mathrm{T}}\boldsymbol{\alpha}\ddot{\mathbf{\Phi}} + \ddot{\mathbf{\Phi}}_{\ddot{\mathbf{q}}}^{\mathrm{T}}\boldsymbol{\kappa} = \mathbf{P}\left(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*}\right) + c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\ddot{\mathbf{\Phi}} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\kappa} =$$

$$\mathbf{P}\left(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^{*}\right) + c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\left(\mathbf{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{t}\right) + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\kappa} = \mathbf{0} \tag{2.76}$$

In (2.76), the following kinematic relations were used.

$$\ddot{\mathbf{\Phi}} = \mathbf{\Phi}_{\mathbf{q}}\ddot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\mathbf{\Phi}}_{t} \tag{2.77}$$

$$\ddot{\mathbf{\Phi}}_{\ddot{\mathbf{q}}} = \frac{\partial \ddot{\mathbf{\Phi}}}{\partial \ddot{\mathbf{q}}} = \mathbf{\Phi}_{\mathbf{q}} \tag{2.78}$$

Expression (2.76) is a nonlinear system of equations that can be solved using the fixed-point iteration.

$$\left(\mathbf{P} + c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\mathbf{\Phi}_{\mathbf{q}}\right)\ddot{\mathbf{q}}_{i+1} = \mathbf{P}\ddot{\mathbf{q}}^{*} - c\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\mathbf{\Phi}_{t} - \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\kappa}_{i+1} \tag{2.79}$$

$$\boldsymbol{\kappa}_{i+1} = \boldsymbol{\kappa}_{i} + c\boldsymbol{\alpha}\ddot{\mathbf{\Phi}} \tag{2.80}$$

There are a number of possibilities to select the projection matrix P and the constant c. This selection strongly affects the behavior of the projections. In the case of the velocity projections, the selection has a useful physical meaning in terms of energy dissipation, as it was described in [41]. There are two interesting options.

1. The original projections of Bayo and Ledesma [7]: $\mathbf{P} = \mathbf{M}$, $c = 1$.

   It was proved in [41] that this selection introduces unconditional dissipation to any incompatible velocity field in the velocity projections, which produces a very stable constraints behavior.

2. The modified projections of Cuadrado et.al. [24]: $\mathbf{P} = (1 - \delta_m)\, \mathbf{M} + (1 - \delta_f)\, \gamma h \mathbf{C}_{n+1} + (1 - \delta_f)\, \beta h^2 \mathbf{K}_{n+1}$, $c = (1 - \delta_f)\, \beta h^2$. This choice makes the coefficient matrix of systems (2.71) and (2.79) equal to the tangent matrix (2.61) and therefore the previous factorization of the tangent matrix, carried out to solve the system (2.57) can be used.

2.4. Penalty matrix

Up to this point, different solving methods for the equations of motion have been presented. When using the Augmented Lagrangian method, which combines the Lagrange multiplier and the penalty method, a single scalar constant α was used as the penalty factor. There are, nevertheless, occasions where replacing this constant by a penalty matrix leads to more efficient schemes.

For mechanical problems involving large differences in magnitude between the masses of their bodies, convergence problems can arise in the iterative process. In the original definition of the Augmented Lagrangian method, constraint forces inserted directly into the equations of motion. An excessively high value for the penalty factor worsens the convergence ratio because the approximation steps to the solution can be much larger than the required ones. Thus, the iteration oscillates near the real solution, but the approaching process can be very slow or even it can never reach convergence. Using the same penalty factor for bodies with very heterogeneous mass values can show this effect for light bodies, since the used value is excessively high for them.

A technique consisting in the use of a penalty matrix can alleviate this issue. This matrix holds individual penalty factors for each constraint equation. The method makes it possible to scale each constraint force $\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \boldsymbol{\lambda}^* + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \boldsymbol{\alpha} \mathbf{\Phi}$ by means of a possibly different weighting proportional factor. A good choice for a weighting finding process that computes the penalty factors can be based in the ratio of each body mass value with respect to the mean value of all the bodies masses. Therefore, all the factors related to constraints involving a certain body can be scaled according its mass value.

$$\overline{m} = \frac{\sum_{i=1}^{n} m_i}{n} \tag{2.81}$$

$$\boldsymbol{\alpha} = \begin{pmatrix} \alpha_g \frac{m_1}{\overline{m}} \\ \alpha_g \frac{m_2}{\overline{m}} \\ \alpha_g \frac{m_3}{\overline{m}} \\ \vdots \\ \alpha_g \frac{m_n}{\overline{m}} \end{pmatrix}^{\mathrm{T}} \mathbf{I} \tag{2.82}$$

where α_g is the global penalty factor, m_i the mass value of the i^{th} object, and $\mathbf{I}$ an $n \times n$ identity matrix, being n the number of bodies in the system. Using the penalty matrix $\boldsymbol{\alpha}$ into the equations of motion (2.10):

$$\mathbf{M}\ddot{\mathbf{q}} + \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\lambda}^{*} + \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\alpha}\boldsymbol{\Phi} = \mathbf{Q}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.83}$$

2.5. Flowchart

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

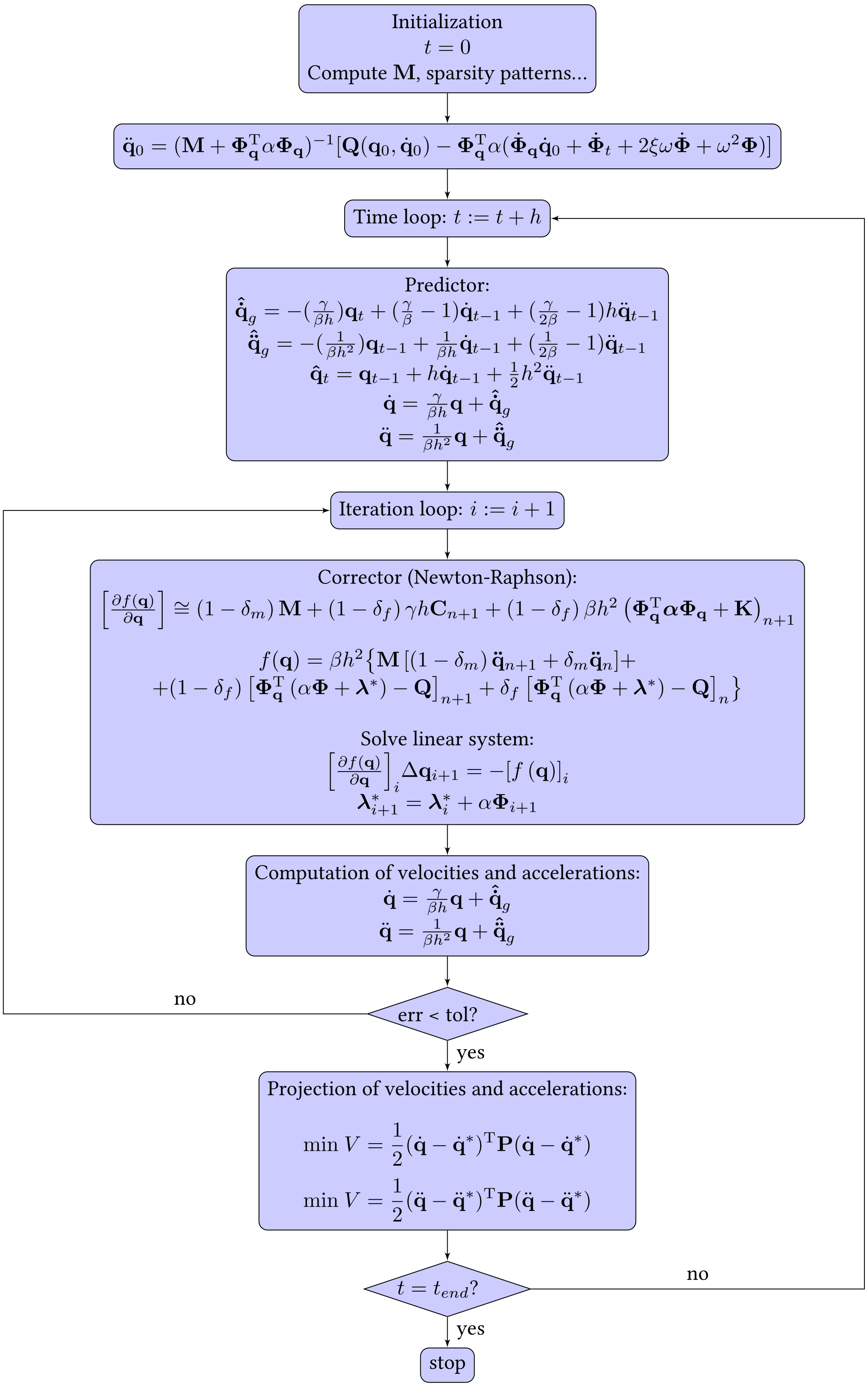

Initialization
$t = 0$
Compute $\mathbf{M}$, sparsity patterns...
$\ddot{\mathbf{q}}_0 = (\mathbf{M} + \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha \mathbf{\Phi}_{\mathbf{q}})^{-1} [\mathbf{Q}(\mathbf{q}_0, \dot{\mathbf{q}}_0) - \mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \alpha (\dot{\mathbf{\Phi}}_{\mathbf{q}} \dot{\mathbf{q}}_0 + \dot{\mathbf{\Phi}}_t + 2\xi\omega\dot{\mathbf{\Phi}} + \omega^2 \mathbf{\Phi})]$
Time loop: $t := t + h$
Predictor:
$\hat{\dot{\mathbf{q}}}_g = -(\frac{\gamma}{\beta h})\mathbf{q}_t + (\frac{\gamma}{\beta} - 1)\dot{\mathbf{q}}_{t-1} + (\frac{\gamma}{2\beta} - 1)h\ddot{\mathbf{q}}_{t-1}$
$\hat{\ddot{\mathbf{q}}}_g = -(\frac{1}{\beta h^2})\mathbf{q}_{t-1} + \frac{1}{\beta h}\dot{\mathbf{q}}_{t-1} + (\frac{1}{2\beta} - 1)\ddot{\mathbf{q}}_{t-1}$
$\hat{\mathbf{q}}_t = \mathbf{q}_{t-1} + h\dot{\mathbf{q}}_{t-1} + \frac{1}{2}h^2\ddot{\mathbf{q}}_{t-1}$
$\dot{\mathbf{q}} = \frac{\gamma}{\beta h}\mathbf{q} + \hat{\dot{\mathbf{q}}}_g$
$\ddot{\mathbf{q}} = \frac{1}{\beta h^2}\mathbf{q} + \hat{\ddot{\mathbf{q}}}_g$
Iteration loop: $i := i + 1$
Corrector (Newton-Raphson):
$\left[\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\right] \cong (1 - \delta_m)\mathbf{M} + (1 - \delta_f)\gamma h \mathbf{C}_{n+1} + (1 - \delta_f)\beta h^2 \left(\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}} \boldsymbol{\alpha} \mathbf{\Phi}_{\mathbf{q}} + \mathbf{K}\right)_{n+1}$
$f(\mathbf{q}) = \beta h^2 \{\mathbf{M}[(1 - \delta_m)\ddot{\mathbf{q}}_{n+1} + \delta_m \ddot{\mathbf{q}}_n] +$
$+(1 - \delta_f)\left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}(\alpha\mathbf{\Phi} + \boldsymbol{\lambda}^*) - \mathbf{Q}\right]_{n+1} + \delta_f \left[\mathbf{\Phi}_{\mathbf{q}}^{\mathrm{T}}(\alpha\mathbf{\Phi} + \boldsymbol{\lambda}^*) - \mathbf{Q}\right]_n\}$
Solve linear system:
$\left[\frac{\partial f(\mathbf{q})}{\partial \mathbf{q}}\right]_i \Delta\mathbf{q}_{i+1} = -[f(\mathbf{q})]_i$
$\boldsymbol{\lambda}_{i+1}^* = \boldsymbol{\lambda}_i^* + \alpha\mathbf{\Phi}_{i+1}$
Computation of velocities and accelerations:
$\dot{\mathbf{q}} = \frac{\gamma}{\beta h}\mathbf{q} + \hat{\dot{\mathbf{q}}}_g$
$\ddot{\mathbf{q}} = \frac{1}{\beta h^2}\mathbf{q} + \hat{\ddot{\mathbf{q}}}_g$
no
err < tol?
yes
Projection of velocities and accelerations:
$\min V = \frac{1}{2}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{P}(\dot{\mathbf{q}} - \dot{\mathbf{q}}^*)$
$\min V = \frac{1}{2}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*)^{\mathrm{T}}\mathbf{P}(\ddot{\mathbf{q}} - \ddot{\mathbf{q}}^*)$
$t = t_{end}$?
no
yes
stop

Figure 2.3: Flowchart for the Augmented Lagrangian method.

Chapter 3

Contact Models

The treatment of contact forces is a key issue in many applications involving multibody systems with eventual impacts or permanent contacts between bodies. One of the most frequent effect is the contact phenomenon between the mechanism's parts and the environment. Contact is a huge topic that, unfortunately, cannot be solved at an arbitrary degree of precision, but always implies some kind of simplified — although realistic — model use. In a real-time simulation, the constraints are even higher, given the small available amount of time to perform the computations. The main tasks in which a contact characterization system is divided are two: the detection of colliding bodies and the computation of the forces that will try to mimic the real phenomenon as faithfully as possible.

In this chapter, some methods are presented for obtaining sets of forces that will make the objects behave as if they were interfering into each others' motion. This includes the computation of the normal forces that try to avoid or at least minimize the inter-penetration between each pair of solids, and their friction counterparts, which account the resistance force appearing from the sliding motion over their surfaces.

If the application to be designed has interactive requirements, such as a human-in-the-loop simulator, this treatment has to be even more careful because the real-time requirements impose firm constraints on the integration time-step size. Additionally, if a implicit integrator is used, its maximum number of iterations are also limited in order to fulfill the timing requirements. In addition to the efficiency considerations, the simulation has to be stable and robust enough along all the range of possible operations of the system, as well as reproduce the behavior of the real system with an acceptable precision.

To solve the impact problem in multibody systems composed of rigid bodies, the methods can be divided into two families [74, 35, 36]: the discontinuous and the continuous approaches. The rigid body assumption made here means that the bodies are supposed to be rigid and only very small local deformations are required to generate very large contact pressures [68]. The discontinuous approaches assume that the impact occurs instantaneously and changes the momenta balances of the system instantaneously, see e.g. [28, 29]. In [94], the impact is considered also discontinuous, but a fast time scale, which considers the duration of the contact and the flexibility of the colliding bodies, is used to compute the coefficient of restitution to feed back the multibody system equations. On the other hand, the continuous approaches are based on regularized-force models that relate the force and deformation of the bodies in collision [67, 74], or based on unilateral constraints techniques that avoid the penetration between bodies [77, 88, 10, 34]. In applications in which permanent contacts or at least contacts of a significant duration are expected to occur, continuous methods are needed. The continuous methods based on regularized forces include a number of viscoelastic and viscoplastic models (see e.g. [36, 68, 16, 48]).

Between the large number of existent formulations of the equations of motion (see e.g. [69]), the penalty and augmented Lagrangian formulations [6, 7] are characterized by transforming the constraints into forces proportional to the constraints violation. This technique, used along this work, is similar and compatible to that of the continuous-force models for normal contact, which relate the force and deformation of the bodies in contact to avoid the penetration between them.

It is worth to mention that, up to these days, no universally accepted model has been developed for the friction force between bodies under dry conditions. The Coulomb's friction law is the most simple model but has the problem that, when used along with continuous normal force models, the gradient of the force at null tangential velocity is infinite. This fact is unacceptable from the numerical point of view, since the motion has to be solved in discrete time-steps and it is not possible to deal with an infinite gradient at null velocity in these conditions (see e.g. [32]). The solution is to avoid the discontinuity of the Coulomb's model while maintaining the physical characteristics of the friction phenomenon important for the considered application to deal with [36].

Related to the contact models, there are two difficult problems to address in real-time applications, especially when using constant integration time-step, which is the case here. The first one is the fact that the contact takes place in a limited, and sometimes very reduced, number of time steps, so that the algorithm has to be robust enough to overcome hard impacts; the second one was mentioned before and is related to the stability of the friction forces at low velocities and the transition between slipping and sticking.

The contact forces approach proposed for this work comprises two different models: the normal force model and the tangential force model. The two models are presented separately in subsequent sections. The tangential model is an original contribution of this work while the normal model is completely taken from previous works. As it will be described later, in the human-in-the-loop application tackled in this document, the multibody model studied is divided into primitive objects (in the majority of the cases, spheres) for contact detection purposes, which interact with CAD environments composed of triangular meshes. Under these circumstances, all the contacts can be approximated as contacts between primitives and approximated planes at the surface of the bodies. For simplicity, the case of spheres against plane surface bodies is the only case that will be explained here, but the generalization is straightforward.

3.1. Normal contact

3.1.1. Hertz-type models

In order to choose the normal force model, some tests were done with several continuous viscoelastic models, like the Hunt-Crossley model [67], the Lankarani-Nikravesh model [75], and the Kelvin-Voight mode [113]. The results shown by the Hunt-Crossley and Lankarani-Nikravesh models were similar and very satisfactory while the Kelvin-Voight model suffered from a lack of dissipation in hard impacts that must be solved in few time steps. Finally, the normal force model chosen for this work was the Hunt-Crossley model [67]. The model is suited to collisions between massive solids for which the assumption of quasi static contact holds and it can be supposed that the deformation is limited to a small region of the colliding bodies while the remainder of them are assumed to be rigid. The expression for the normal force, after some calculations, has the following form,

$$\mathbf{F}_n = k_n \, \delta^e \left(1 + \frac{3\,(1-\epsilon)}{2} \frac{\dot{\delta}}{\dot{\delta}_0} \right) \mathbf{n} \tag{3.1}$$

where k_n is the equivalent stiffness of the contact and depends on the shape and material properties of the colliding bodies, e is the Hertz's exponent, $\delta = R_{sph} - \|\mathbf{p}_{center} - \mathbf{p}_{contact}\|$ is the indentation, $\dot{\delta}$ its temporal derivative, $\dot{\delta}_0$ is the relative normal velocity between the colliding bodies when the contact is detected, ϵ is the coefficient of restitution, and $\mathbf{n}$ is the direction of the force (see Figure 3.1). The subscript "*n*" comes from "*normal*".

The value of k_n can be calculated for general colliding paraboloids but, as was mentioned before, for this explanation all the contacts will be considered as contacts between spheres and plane surface bodies in which case the expression for the stiffness can be expressed by (see for example [46]),

$$k_n = \frac{4}{3\,(\sigma_{sph} + \sigma_{pln})} \sqrt{R_{sph}} \tag{3.2}$$

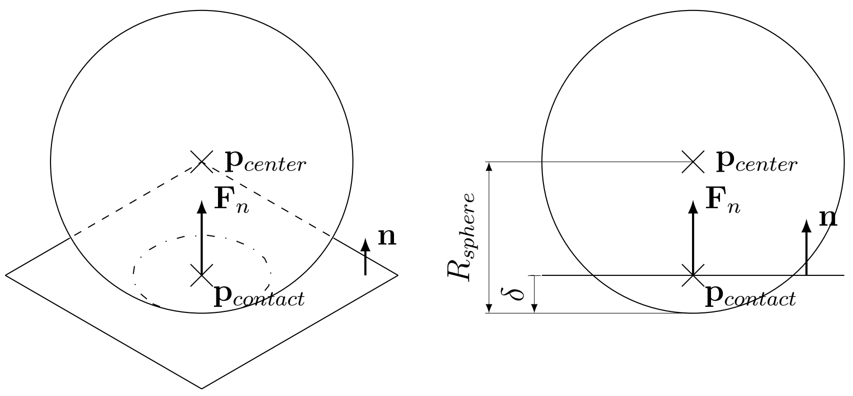


Figure 3.1: Normal contact between sphere and plane: isometric and front views.

where R_{sph} is the radius of the sphere in contact with the plane, and the material parameters of the sphere and plane, σ_{sph} and σ_{pln} are given by,

$$\sigma_{sph} = \frac{1-\nu_{sph}^2}{E_{sph}}; \quad \sigma_{pln} = \frac{1-\nu_{pln}^2}{E_{pln}} \tag{3.3}$$

and ν and E stand for the Poisson's ratio and the Young's modulus of each one of the two materials, represented by the sphere and plane.

It was explained in Chapter 2 that the formulation chosen for this work uses implicit integration and Newton-Raphson iteration and therefore, the forces in the multibody system contribute to the tangent matrix (2.20) by means of the stiffness and damping matrices (2.21) and (2.22). For the Hunt-Crossley model presented here, the contribution to these matrices includes the following derivatives.

$$\mathbf{K}_n = -\frac{\partial \mathbf{F}_n}{\partial \mathbf{p}_{center}} = -\mathbf{n}\left(e\,k_n\delta^{e-1}\left(1+\frac{3\,(1-\epsilon)}{2}\frac{\dot{\delta}}{\dot{\delta}_0}\right)\right)\mathbf{n}^{\mathrm{T}} \tag{3.4}$$

$$\mathbf{C}_n = -\frac{\partial \mathbf{F}_n}{\partial \dot{\mathbf{p}}_{center}} = -\mathbf{n}\left(k_n\delta^{e}\left(\frac{3\,(1-\epsilon)}{2\dot{\delta}_0}\right)\right)\mathbf{n}^{\mathrm{T}} \tag{3.5}$$

The previous expressions are not yet the contributions to the tangent matrix of (2.16), because depending on the natural coordinates chosen in the construction of the multibody model, it is usually necessary to perform additional substitutions and derivatives to convert the previous expressions into derivatives of generalized forces with respect to generalized coordinates.

3.2. Tangential force model

The tangential force model developed for the friction force is based on Coulomb's law including stiction. Moreover a viscous term is added to the dry friction force. The general form of this force is the following,

$$\mathbf{F}_t = \kappa\, \mathbf{F}_{stic} + (1-\kappa)\, \mathbf{F}_{slide} - \mu_{visc}\mathbf{v}_t \tag{3.6}$$

In the previous expression, the first two terms constitute the dry friction, while the third term accounts for the viscous friction. For the smooth transition between sticking and slipping the dry friction force is divided in two components coupled by a smooth function, following the ideas proposed in [48]. The subscript "*t*" comes from "*tangential*".

In (3.6), μ_{visc} is the viscous damping coefficient, $\mathbf{F}_{stic}$ and $\mathbf{F}_{slide}$ are the components of the stiction and slipping forces, κ is a smooth function of the tangential velocity, $\mathbf{v}_t$, which is defined in terms of the central point of the contact region, $\mathbf{p}_{contact}$, and the normal vector at the contact, $\mathbf{n}$, as follows.

$$\mathbf{v}_t = \dot{\mathbf{p}}_{contact} - \left(\mathbf{n}^{\mathrm{T}}\, \dot{\mathbf{p}}_{contact}\right) \mathbf{n} \tag{3.7}$$

The mentioned function, κ, has to match the following conditions,

$$\kappa = \left\{ \begin{array}{ll} 0; & \|\mathbf{v}_t\| >> v_{stic} \\ 1; & \|\mathbf{v}_t\| = 0 \end{array} \right\} \tag{3.8}$$

where v_{stic} is a parameter of the model accounting for the velocity of the stick-slip transition. A good choice for the transition function κ was given in [48] and has the following form.

$$\kappa = \mathrm{e}^{-\left(\mathbf{v}_t^{\mathrm{T}}\mathbf{v}_t\right)/v_{stic}^2} \tag{3.9}$$

Equation (3.6) showed that the total force is composed of three contributions: the sliding dry friction force at high velocities, the stiction force at low velocities and the viscous friction force. The stiction force will be considered by means of viscoelastic elements acting between the colliding bodies, called bristles. The expressions of the sliding and stiction forces are given by (3.10) and (3.11) (see Figure 3.2).

$$\mathbf{F}_{slide} = \left\{ \begin{array}{ll} 0; & \|\mathbf{v}_t\| = 0 \\ -\mu_{din}\, \|\mathbf{F}_n\|\, \dfrac{\mathbf{v}_t}{\|\mathbf{v}_t\|}; & \|\mathbf{v}_t\| > 0 \end{array} \right\} \tag{3.10}$$

$$\mathbf{F}_{stic} = \left\{ \begin{array}{ll} 0; & s = 0 \\ \dfrac{f_{stic}^m}{s}\left(\mathbf{I}_3 - \mathbf{n}\mathbf{n}^{\mathrm{T}}\right)\left(\mathbf{p}_{contact} - \mathbf{p}_{stic}\right); & s > 0 \end{array} \right\} \tag{3.11}$$

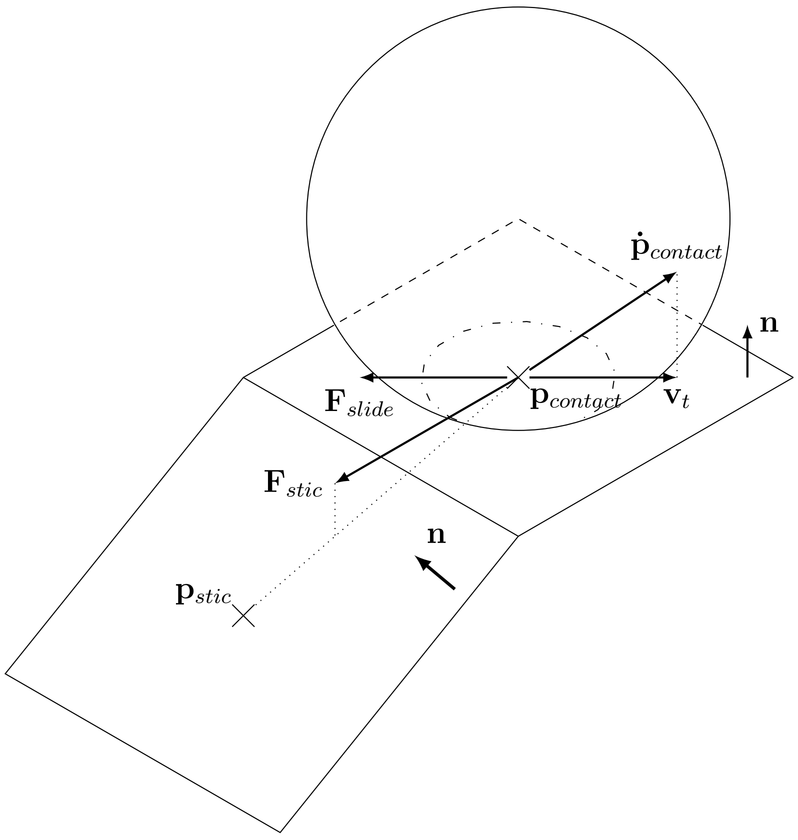


Figure 3.2: Tangential contact between sphere and plane.

being μ_{din} the friction coefficient under dynamic conditions, $s = ||\mathbf{p}_{contact} - \mathbf{p}_{stic}||$ the deformation of the bristles, with $\mathbf{p}_{stic}$ the stiction point, which was initially the central point of the contact region in the instant in which the contact began; $\mathbf{I}_3$ is the identity matrix of size 3×3; f^m_{stic} is the function that represents the behavior of the bristles,

$$f^m_{stic} = -k_{stic}\, s - c_{stic}\, \dot{s} \tag{3.12}$$

being k_{stic} and c_{stic} the stiffness and damping coefficients of the stiction model. Nevertheless there is a limiting value for the stiction force.

$$||\mathbf{F}_{stic}|| \leq \mu_{st}||\mathbf{F}_n|| \tag{3.13}$$

In (3.13), μ_{st} is the friction coefficient under static conditions which is, in general, higher than the dynamic friction coefficient. In case this limit is exceeded, and (3.13) is not fulfilled, there are two consequences: first, the equation (3.12) is not valid anymore and the equation (3.14) holds for the behavior of the bristles; and second, the stiction point has to be updated using (3.15).

$$f^m_{stic} = \frac{-\mu_{st}||\mathbf{F}_n||\; s}{||\left(\mathbf{I}_3 - \mathbf{n}\mathbf{n}^{\mathrm{T}}\right)\left(\mathbf{p}_{contact} - \mathbf{p}_{stic}\right)||} \tag{3.14}$$

$$\mathbf{p}_{stic} = \mathbf{p}_{contact} - \left(\frac{\eta_{stic}\, \mu_{st}\; ||\mathbf{F}_n||}{k_{stic}}\right)\frac{\mathbf{v}_t}{||\mathbf{v}_t||} \tag{3.15}$$

The coefficient η_{stic} controls the strain of the bristles when the maximum force is reached. Physically the more reasonable value is $\eta_{stic} = 1$, but small variations with $\eta_{stic} < 1$ can improve the numerical behavior of the model.

The contribution of the tangential force, $\mathbf{F}_t$, to the tangent matrix (2.20) includes the following derivatives.

1. Case $||\mathbf{F}_{stic}|| \leq \mu_{st}||\mathbf{F}_n||$.

$$\mathbf{K}_t = -\frac{\partial \mathbf{F}_t}{\partial \mathbf{p}_{contact}} = \kappa\left(\mathbf{I}_3 - \mathbf{n}\mathbf{n}^{\mathrm{T}}\right)\mathbf{K}^m_{stic} + (1-\kappa)\mu_{din}\frac{\mathbf{v}_t}{||\mathbf{v}_t||}\frac{\partial ||\mathbf{F}_n||}{\partial \mathbf{p}_{contact}} \tag{3.16}$$

where

$$\mathbf{K}^m_{stic} = \frac{1}{s^2}\left(k_{stic} + \frac{f^m_{stic}}{s}\right)\left(\mathbf{p}_{contact} - \mathbf{p}_{stic}\right)\left(\mathbf{p}_{contact} - \mathbf{p}_{stic}\right)^T - \frac{f^m_{stic}}{s}\mathbf{I}_3 \tag{3.17}$$

The damping contribution.

$$\begin{aligned}\mathbf{C}_t = & -\frac{\partial \mathbf{F}_t}{\partial \dot{\mathbf{p}}_{contact}} = \kappa\left(\mathbf{I}_3 - \mathbf{n}\mathbf{n}^{\mathrm{T}}\right)\mathbf{C}^m_{stic} + \\ & (1-\kappa)\left[\mu_{din}\frac{\mathbf{v}_t}{\|\mathbf{v}_t\|}\frac{\partial\|\mathbf{F}_n\|}{\partial \dot{\mathbf{p}}_{contact}} + \frac{\mu_{din}\|\mathbf{F}_n\|}{\|\mathbf{v}_t\|}\left(\mathbf{I}_3 - \frac{\mathbf{v}_t\mathbf{v}_t^T}{\|\mathbf{v}_t\|^2}\right)\frac{\partial \mathbf{v}_t}{\partial \dot{\mathbf{p}}_{contact}}\right] + \\ & \frac{2s}{v^2_{stic}}(\mathbf{F}_{stic} - \mathbf{F}_{slide})\mathbf{v}_t^{\mathrm{T}}\frac{\partial \mathbf{v}_t}{\partial \dot{\mathbf{p}}_{contact}} + \mu_{visc}\frac{\partial \mathbf{v}_t}{\partial \dot{\mathbf{p}}_{contact}}\end{aligned} \tag{3.18}$$

being

$$\mathbf{C}^m_{stic} = \frac{c_{stic}}{s^2}\left(\mathbf{p}_{contact} - \mathbf{p}_{stic}\right)\left(\mathbf{p}_{contact} - \mathbf{p}_{stic}\right)^T \tag{3.19}$$

$$\frac{\partial \mathbf{v}_t}{\partial \dot{\mathbf{p}}_{contact}} = \mathbf{I}_3 - \mathbf{n}\mathbf{n}^{\mathrm{T}} \tag{3.20}$$

2. Case $\|\mathbf{F}_{stic}\| > \mu_{st}\|\mathbf{F}_n\|$.

$$\mathbf{K}_t = \mathbf{0} \tag{3.21}$$

$$\mathbf{C}_t = \mu_{visc}\frac{\partial \mathbf{v}_t}{\partial \dot{\mathbf{p}}_{contact}} \tag{3.22}$$

It is important to note again that the previous expressions of $\mathbf{K}_t$ and $\mathbf{C}_t$ are not yet the contributions to the tangent matrix (2.20), because depending on the natural coordinates chosen in the construction of the multibody model, it is usually necessary to perform additional substitutions and derivatives to convert the previous expressions into derivatives of generalized forces with respect to generalized coordinates.

3.3. Numerical Examples

The formulation with the contact model proposed is tested in two different applications: the first one is the simulation of a spring-mass system with Coulomb's friction, which is an academic problem with known analytic solution; the second application is the well known Bowden and Leben stick-slip experiment.

In all the examples of this work, the contact forces were included by means of the normal force model explained in §3.1 and the tangential force model developed in §3.2.

The mass-spring and Bowden and Leben stick-slip examples (§3.3.1 and §3.3.2) were implemented in Fortran 2003 language.

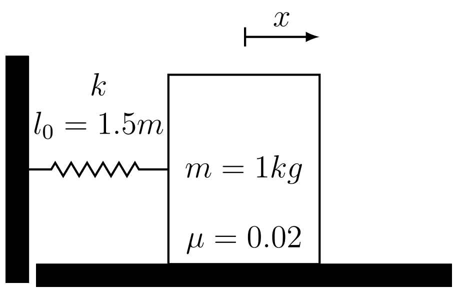


Figure 3.3: Mass-spring system with Coulomb's friction.

3.3.1. Mass-spring system with Coulomb's friction

The first system to be simulated is the mass-spring system with Coulomb's friction shown in Figure 3.3, which is a very simple example with known analytic solution but at the same time interesting, to test the tangential contact model proposed, and to compare it with known theoretical results.

The simulation total time is 13 seconds and the time step $h = 0.01$ seconds. The system undergoes the influence of the gravity forces $g = 9.81m/s^2$. The numerical values of the parameters are: the mass of the block $m = 1\ kg$, the dynamic friction coefficient $\mu_{din} = 0.02$, the static friction coefficient $\mu_{st} = \mu_{din}$, while the viscous friction coefficient $\mu_{visc} = 0$, the natural spring length $l_0 = 1.5\ m$, the stiffness coefficient of the spring is $k = \left\{ \begin{array}{ll} 1\ N/m; & t < 10\ s \\ 10\ N/m; & t \geq 10\ s \end{array} \right\}$, being t the integration time variable. The change on the spring stiffness is motivated to force the stick-slip transition when the simulation time reaches $t = 10\ s$, just before this instant the mass was stuck to the plane and the change in the spring stiffness forces the mass to move.

For this example the actual geometry of the block is neglected and only two contact points are considered, one in each end of the block. The normal and tangential forces are introduced to these points. The block is constrained to move and rotate in the plane of the figure so the system has 3 degrees of freedom.

The parameters of the normal force model have little influence in the response (provided the stiffness of the contacts is sufficient and the restitution coefficient is not close to 1).

The remaining parameters for the tangential contact model described in §3.2 have the following values: $v_{stic} = N\mu gh$, $k_{stic} = m/(Nh)^2$, $c_{stic} = 2\sqrt{k_{stic}m}$, $\eta_{stic} = 1$. The parameter N allows to estimate the rest of parameters of the model and intends to be the number of time steps to stop the mass once the stiction is acting, and is set to $N = 5$ for this application. Excessively low values lead to numerical problems.

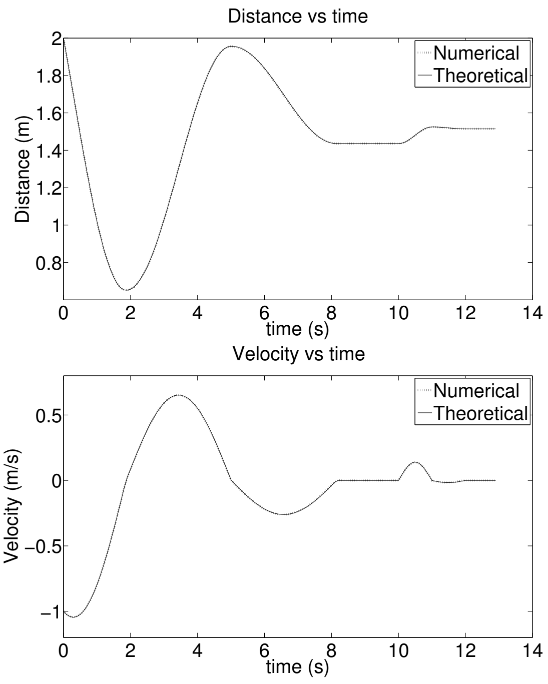


Figure 3.4: Response of the system (block position and velocity): theoretical vs. numerical.

The theoretical responses of the system vs. the numerical responses are shown in Figure 3.4, the magnitudes represented are the block position (spring distance) and the block velocity (spring distance derivative). The coincidence between the theoretical and numerical responses is quite good, and the slip-stick and stick-slip transitions are also satisfactory.

3.3.2. The Bowden and Leben stick-slip experiment

The second system to be simulated is the Bowden and Leben stick-slip experiment. This experiment was first proposed by Bowden and Leben to study the stick-slip process [90] and it was described and solved recently in [48].

The system consists of a mass-spring system similar to that presented in §3.3.1 but, in this case, the block is mounted on a conveyor belt that is moved at a constant speed as shown in Figure 3.5.

The simulation total time is 120 seconds and the time step $h = 0.001$ seconds. The system undergoes the influence of the gravity forces $g = 10m/s^2$. The numerical values of the parameters are: the mass of the block $m = 1\ kg$, the dynamic friction coefficient $\mu_{din} = 0.1$, the static friction coefficient $\mu_{st} = 0.15$, the viscous friction coefficient $\mu_{visc} = 0.1$, the natural spring length $l_0 = 1.5\ m$,

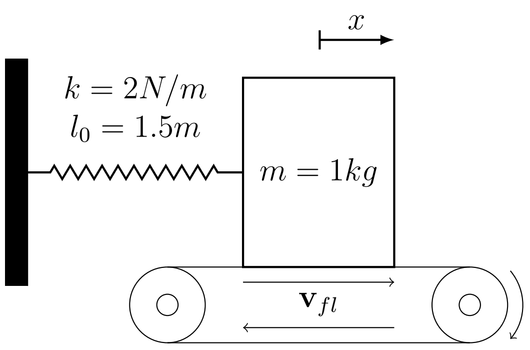


Figure 3.5: The Bowden and Leben stick-slip experimental apparatus.

the stiffness coefficient of the spring is $k = 2N/m$ and the velocity of the conveyor belt is $\mathbf{v}_{fl} = 0.05m/s$ ("*fl*" from floor).

Like in the previous example, the actual geometry of the block is neglected and only two contact points are considered, one in each end of the block and the contact forces are introduced to these points.

The remaining parameters for the tangential contact model described in §3.2 have the following values: $v_{stic} = 0.001m/s, k_{stic} = 10^5 N/m, c_{stic} = \sqrt{10^5} Ns/m,$ $\eta_{stic} = 1$.

The responses of the system are shown in Figure 3.6, the magnitudes represented are the block position (spring distance), the block velocity (spring distance derivative) and the magnitude of the friction force.

Note in Figure 3.6b that the block is stuck to the conveyor belt until the spring force equals the maximum friction force available under static conditions ($\mu_{st}mg$). At this moment the block begin to slide and the maximum friction force available, abruptly drops to the maximum force under dynamic conditions ($\mu_{din}mg$). Moreover, due to the relative velocity between the block and the conveyor belt, the viscous friction acts ($\mu_{visc}v_t$), which is responsible of the small round peaks that can be observed in Figure 3.6c). Finally, the block sticks again to the conveyor belt with a stiction force that is much lower than the maximum available.

The response of the model presented here is very similar to that presented in [48], except by the fact that the model presented here does not take into account the dwell-time dependency of the stiction force.

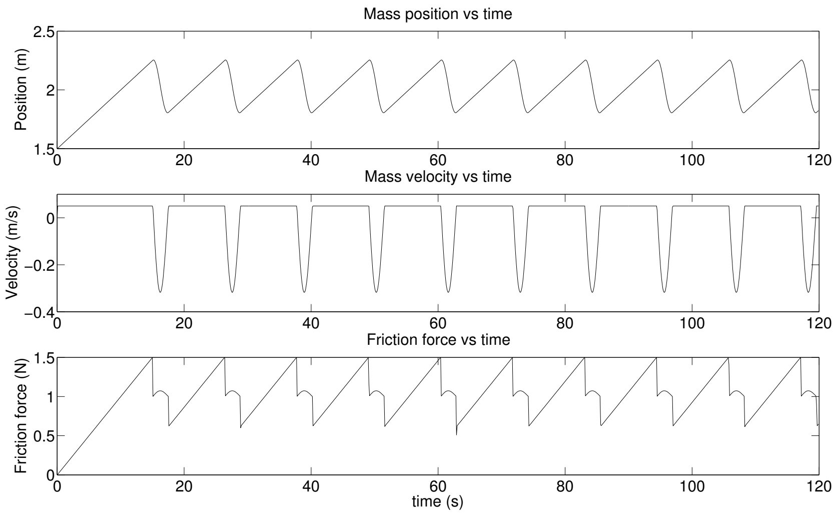


Figure 3.6: Response of the system: a) block position, b) block velocity and c) friction force.

3.4. Terrain model

In this section, a terrain contact model is presented. The model tries to represent the behavior of the interaction of a solid object and a piece of terrain. It is specially focused on representing excavation maneuvers in machinery simulators. That way, it can approximate the amount of material removed from a terrain patch, and generate a dragging force, opposed to the movement of the excavation bucket. An implementation example is shown in chapter 6.

The soft soil is modeled as a terrain mesh. During the excavation process, the bucket penetrates the terrain mesh. A ray-casting method is used to compute the intersection area between the bucket admission and the mesh representing the terrain surface; this area is integrated using the velocity component of the bucket normal to the area, in order to compute the volume and weight of earth loaded by the bucket in each time step.

In addition, the algorithm diminishes the z-coordinates of the points from the removable terrain mesh that have entered inside the bucket. The algorithm provides a reasonable estimation of the loaded weight and a realistic visualization of the process.

The unloading process is simulated in a similar way: when the front of the bucket surpasses a predefined critical angle, a flow of material is cast from it; the dropping process of the material is simulated, the exact instant and the location where this flow contacts the terrain mesh or other object (e.g. a truck) is calculated. If this location does not belong to the terrain mesh, a generic flat mesh is created at that position. The flow of material is used to modify the height field of the mesh, increasing its z-coordinates using a Gaussian distribution to distribute the material randomly around the intersection point.

3.4.1. Implementation

The filling of a excavator bucket is a complex granular flow problem. The majority of the numerical models that try to consider the different mechanisms involved in the filling process, are not suitable for real-time applications because they were developed for design optimization, and they are too heavy for real-time purposes. On the other hand, there are also some analytic models which are good for simple geometries and better suited to real-time simulations. This is the option chosen here.

In order to calculate the bucket digging force, different types of soil failure mechanisms have to be considered [70]: rigid-brittle type of failure and flow failure. In brittle failure, blocks of soil are periodically separated from the soil mass, and the force on the tool is of periodic nature in brittle failure. Speed does not affect the shear strength under the conditions of brittle soil failure.

In this work, the brittle type of failure will be neglected; being possible, thus, to develop a simplified force model in terms of a given bucket depth and the soil volume flow towards the bucket. The simplified expression to calculate the digging force is the following,

$$\mathbf{F}_{dig} = -\mu_{dig}(\sigma_1 + \sigma_2 V_b^m) d_b^n \mathbf{v}_t - \mu_{cp} d_b^n \mathbf{v}_n \tag{3.23}$$

Where $\mathbf{F}_{dig}$ is the digging force, V_b is the volume of material inside the bucket, d_b is the bucket depth and σ_1, σ_2, μ_{dig}, μ_{cp}, m and n are parameters of the model; $\mathbf{v}_d$ is the velocity of the bucket's teeth, $\mathbf{v}_t$ is the projection of $\mathbf{v}_d$ onto the excavation direction and $\mathbf{v}_n = \mathbf{v}_d - \mathbf{v}_t$ (see Figure 3.7a).

The weight of the soil inside the bucket is directly calculated from the volume V_b:

$$\mathbf{P}_{soil} = \rho V_b \mathbf{g} \tag{3.24}$$

3.4.2. Charge material maneuver

In the calculation of V_b, some geometrical and kinematic properties of the bucket object have to be considered. The volume is computed from the incoming flow of material given the bucket motion. That flow is the filled area of the bucket,

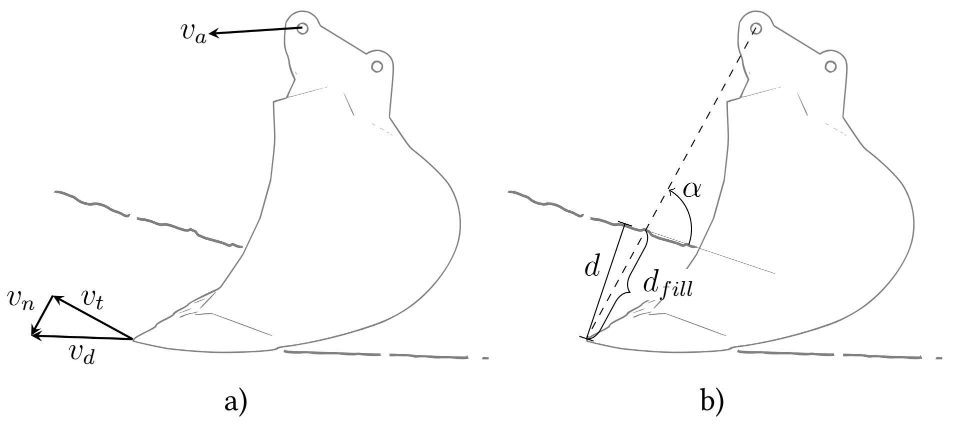


Figure 3.7: Parameters for the soil excavation force and bucket filling models.

given its velocity. Therefore, the amount of material entering into the bucket and its velocity have to be determined.

The part of the bucket aperture that is filled with material is computed from the position of the bucket's teeth regarding the soil. The depth is used to compute a filling level with the bucket orientation (Figure 3.7b):

$$d_{fill} = \frac{d}{\sin \alpha} \tag{3.25}$$

As shown, only the orientation of the bucket on one plane is used. This is not generally exact when the slope in the perpendicular direction changes greatly along the bucket's width. To overcome that problem, bucket volume computations are sliced into several parts that are added at the end, thus approaching to more exact results.

The velocity profile in the plane for each point of the bucket aperture can be computed given the values for any two of its points. The multibody software can provide the velocities of the bucket at the point of any of its teeth, and also at its hinge. A linear function can be found as shown in (3.26). There, $\mathbf{v}_h$ is the velocity of the hinge point at the bucket, and r is the radius of the bucket aperture: therefore its length is $2r$.

$$\mathbf{v}(x) = \mathbf{v}_d + \frac{\mathbf{v}_d - \mathbf{v}_h}{2r} x \tag{3.26}$$

Assuming the mean velocity for the flow is the one found at the point $x = \frac{d_{fill}}{2}$, its value is

$$\mathbf{v}\left(\frac{d_{fill}}{2}\right) = \mathbf{v}_d + \frac{d_{fill}}{4r}(\mathbf{v}_d - \mathbf{v}_h) \tag{3.27}$$

Finally, the mass increment per time step and by slice i is

$$\Delta V_i = \rho \mathbf{v}_{\frac{d_{fill}}{2}} A = \rho(\mathbf{v}_d + \frac{d_{fill}}{4r}(\mathbf{v}_d - \mathbf{v}_h))d_{fill}w_s \tag{3.28}$$

with w_s being the width of a slice.

Chapter 4

Contact Detection

Multibody systems are often found as an embedded component into larger systems, which benefit from the available computed motion information that this tool provides. *HiL* systems, a term which can stand for *Human in the Loop* or *Hardware in the Loop*, are common cases where multibody techniques are valuable. A multibody system calculates the dynamic state of a mechanical system from its current configuration state and the forces that are applied to the mechanism. Occasionally, it is needed to modify directly the coordinates of the multibody system to alter its configuration, while most of the time, this is achieved by applying external forces to the mechanism. It is very usual to have to introduce forces in the model as a result of the occurrence of contacts between one of the mechanism's parts and any other part or object present in the simulation.

One of the strengths of a multibody simulation is to be able to study or replicate the behavior of a system, including collision and contact events. The models used have to be able to deliver a realistic response during and until the end of the event.

Some research fields do not need that complexity because contact events are to be avoided. Therefore, it could be enough to detect if a contact event happens or not. In robotics, for example, self-collisions and obstacle circumvention are studied in order for a device to accomplish its tasks, while maximizing its work space. Collisions are not allowed during the operation, so it is enough to test if they exist at each time instant. However, multibody simulations are just often performed in order to study processes where contacts are an essential part of the problem. Tools and machines often work by grasping, pushing, lifting other objects, and those actions are characterized by contacts. In addition to being able to detect contacts, the model should be sophisticated enough to provide the parameters needed to compute the resultant forces.

Characterizing a contact as a multibody input event is a challenge by itself, since there is no trivial way of interpreting it as force or a constraint. Actually, contact phenomena leads to deformation of the surface of bodies involved, so the rigid body approach can be regarded as a rough approximation. Nowadays it is not possible to execute complex contact models in real time. Therefore, some simplified models have been developed in order to be able to compute reasonable approximations for the contact problem.

The goal is to compute the forces that would appear as a result of existent collisions, so the bodies' motion is physically correct. The quality of the force model can be determined by the energy balance before and after the collision event, and the degree of interpenetration between colliding bodies. Two popular approaches derive from those requirements: ensuring that there is no interpenetration between the bodies through the use of constraints, or using a force model which encompasses the elastic deformation from the contact, characterized by the interpenetration magnitude.

4.1. Unilateral contacts

Modeling contacts with constraints is done by imposing them when a collision happens — therefore avoiding interpenetration — and removing them as soon as it is detected that they could prevent any pair of bodies to separate. These constraints are called *unilateral constraints* since they only allow motion in one direction — when the bodies are moving away — and thus, they are active only if certain conditions are met.

Unilateral constraints are activated when a collision is detected between a pair of bodies, based on their geometry definition. They will be later released when the reaction force that the constraints are imposing would attract the bodies instead of separating them.

In the simplest cases, those activation-deactivation checks can be handled in an individual basis, with a simple conditional algorithm, where constraints are created as soon as a pair of bodies is known to be in contact, and released when the constraint's reaction force of a body is pointing towards the surface of the opposite body. This method usually allows to easily consider contact phenomena in a simulation by means of a simple contact model. Usually, the only physical parameters needed for the contact model are the coefficient of restitution, $\epsilon = \frac{\mathbf{v}_{out}}{\mathbf{v}_{in}}$, and the coefficient of friction, $\mu = \frac{\mathbf{F}_t}{\mathbf{F}_n}$.

Unfortunately, when dealing with multiple, simultaneous contacts, the activation or deactivation of any individual constraint cannot be considered in isolation. Any of the reaction forces depends on the balance of the rest of the forces applied into the system, including the rest of the reaction forces. Thus, a naive and inefficient method of computing the correct force state of the system would be to check all possible combinations of activation and deactivation of collision

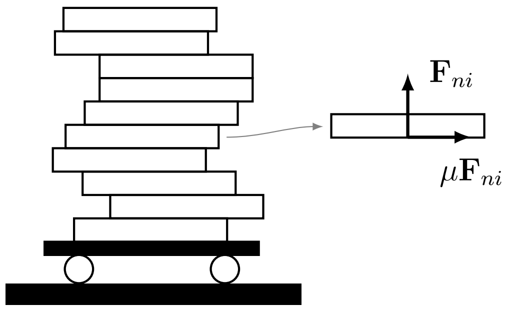


Figure 4.1: The dishes problem.

constraints, until a feasible set is found.

This problem is even more complex when considering friction effects into the system. An additional configuration state is needed for every reaction, given that a body can be either sticking or sliding over its contacting surface. Additional constraints must be also applied if the body should stick to the surface. Those constraints prevent movements in the tangential direction of the surface. Nevertheless, the reaction force imposed by those constraints must be also checked in order not to overcome the friction resistance, that is $\mathbf{F}_t \leq \mu\mathbf{F}_n$. Should this happen, the constraint must be removed in order to allow sliding motion.

A very illustrative example is suggested by [44] and shown in Figure 4.1. The problem consists in determining the motion for a stack of dishes that are based upon a moving floor. At every time step, constraints must be enforced in order to take into account the normal reaction forces between each dish and its neighbors, $\mathbf{F}_{ni}$. Additional constraints must be imposed if any pair of dishes should not have relative motion because of the opposition of the friction forces, $\mu\mathbf{F}_{ni}$. Since each new body increases the potential state count by three, the number of possible constraint combinations of the problem is 3^n, where n is the total number of bodies. As a result, the dimension of the combining problem grows exponentially, and more efficient approaches are needed in order to find the correct constraint configuration.

Glocker [44] proposes to embed the motion equations into a *Linear Complementarity Problem* [21], where unilateral constraints can be stated into the problem definition, and thus solved as a whole. The general form of a LCP is shown in (4.1). Given a matrix $\mathbf{M}$ and a vector $\mathbf{q}$, vectors $\mathbf{w}$ and $\mathbf{z}$ must be found so they satisfy the main expression, while fulfilling the condition that:

- All their components are non-negative.
- For each component i of $\mathbf{w}$ and $\mathbf{z}$, at least w_i or z_i is zero.

$$\begin{aligned} \mathbf{w} &= \mathbf{M}\mathbf{z} + \mathbf{q} \\ \mathbf{w} &\geq \mathbf{0}, \mathbf{z} \geq \mathbf{0} \\ \mathbf{w}_i \mathbf{z}_i &= 0 \ \forall i \end{aligned} \tag{4.1}$$

The adaptation of the LCP method to a multibody dynamics formulation leads to a LCP problem of the form

$$\underbrace{\begin{pmatrix} \ddot{\mathbf{g}} \\ \boldsymbol{\lambda}_{H0} \end{pmatrix}}_{\mathbf{w}} = \underbrace{\begin{pmatrix} \mathbf{W}^{\mathrm{T}}\mathbf{M}^{-1}(\mathbf{W}+\mathbf{N}_G) & \mathbf{I}^{\mathrm{T}} \\ \mathbf{N}_H - \mathbf{I} & \mathbf{0} \end{pmatrix}}_{\mathbf{M}} \underbrace{\begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{z} \end{pmatrix}}_{\mathbf{z}} + \underbrace{\begin{pmatrix} \mathbf{W}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{h} + \overline{\mathbf{w}} \\ \mathbf{0} \end{pmatrix}}_{\mathbf{q}} \tag{4.2}$$

$$\begin{pmatrix} \ddot{\mathbf{g}} \\ \boldsymbol{\lambda}_{H0} \end{pmatrix} \geq \mathbf{0}; \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{z} \end{pmatrix} \geq \mathbf{0} \tag{4.3}$$

$$\begin{pmatrix} \ddot{\mathbf{g}} \\ \boldsymbol{\lambda}_{H0} \end{pmatrix}^{\mathrm{T}} \begin{pmatrix} \boldsymbol{\lambda} \\ \mathbf{z} \end{pmatrix} = \mathbf{0} \tag{4.4}$$

where the unknowns of the system, $\ddot{\mathbf{g}}$ and $\mathbf{z}$, represent the relative normal and tangential accelerations of the contact points, and $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}_{H0}$ are the values of the normal and tangential contact forces. The complementarity condition (4.4) imposes that, for each contact, either the bodies are moving away (the "gap" acceleration $\ddot{g}_i > 0$) or else a contact force is applied ($\lambda_i > 0$), but not both simultaneously. Analogous conditions are imposed for the friction phenomenon, where a body can be either stuck ($\lambda_{H0i} > 0$) or sliding in the tangential direction ($z_i > 0$), but not at the same time. The rest of the values in (4.2) are terms derived from the jacobians of the constraints ($\mathbf{W}$ and $\overline{\mathbf{w}}$, $\mathbf{N}_G$ and $\mathbf{N}_H$), or from the mass properties of the bodies, being $\mathbf{M}$ the mass matrix, $\mathbf{I}$ the inertia matrix, and $\mathbf{h}$ a term accounting for the gyroscopic forces.

The disadvantages of using unilateral constraints methods for modeling contacts have both their origin in the nature of the method itself, and in the process of integrating them into existing multibody frameworks.

Solving the motion system with unilateral constraints requires complex computations, either by solving a LCP system as described before, or by means of the transformation of the unilateral constraints into projective equations in order to get a linear equation system. The contact force models are tightly embedded into the formulation of the system, and can render very difficult to use different ones.

Some coordinate configurations can lead numerical singularities for particular positions of the mechanism. This comes from the disregarding of elastic forces in the contact model, which could disambiguate scenarios where there exist more than one satisfactory solution for the force distribution problem.

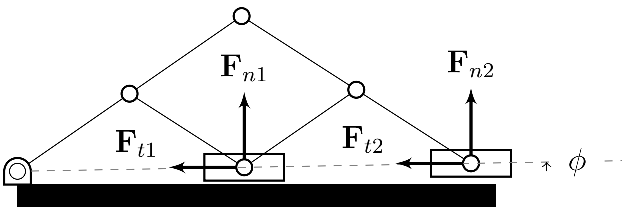


Figure 4.2: Singularity of a pantograph mechanism modeled by a single angular coordinate.

A concrete example is shown again in [44], where a pantograph with two supports rests over a planar surface (Figure 4.2). As the angle ϕ approaches to zero, the contact force distribution over the supports, characterized by normal forces $\mathbf{F}_{n1}$ and $\mathbf{F}_{n2}$, becomes not unique, but undetermined. This can be problematic for finding the values of the friction forces $\mathbf{F}_{t1}$ and $\mathbf{F}_{t2}$, since they depend on the normal forces when using Coulomb's law, see (4.5).

$$\mathbf{F}_n = \mu \mathbf{F}_t \tag{4.5}$$

On the text, a way for overcoming this problem is suggested: using a different set of coordinates that does not expose singularity problems for that specific configuration. A set of Cartesian coordinates (x, y) defining the position of each block of the pantograph could be used. However, as the system size grows, it is more cumbersome to find a minimal singularity-free set of coordinates for the model. Besides that, coordinate selection is a problem that is difficult to solve with a computer, so it will usually have to be carried by an human being.

For generic multibody software, where the kind of mechanism cannot be foreknown, the only choice is to assume a pessimistic set of coordinates to try to minimize the chances of stepping into that problem.

Another disadvantage can be found if a multistep integration scheme is used. Those systems usually present a start up cost, but later stages perform much more efficiently. When any contact occurs, it is necessary to stop the integration process in order to incorporate new constraints. The integration must be also stopped if a contact condition is void, or a stick-slip condition is changed. Should the simulation had frequent collision events, the performance penalty would become very high.

4.2. Force-based contacts

Force models based on body interpenetration are easier to implement, impose less requirements over the multibody system, although they usually require a more careful selection of the stiffness and damping parameters characterizing the contact.

These models are also simplifications, since they assume certain geometric properties, as the Hertz-based models or the elastic foundation model do. Hertz-based models require the local radius of curvature of the surfaces in the contact region to be known. This is a good approximation for solids with sharp surfaces, where the contact region is almost always a locus assimilated to a point or and edge. Nevertheless, for some shapes this model can fall short and some auxiliary fictitious geometries must be created in order to simulate a realistic behavior. An example of this problem is the contact between *conforming* geometries: the contact region is a large area before any deformation occurs, and thus it is no longer possible to model the reaction with only one normal force placed at its center. Consider the case of one box resting over a flat surface, as shown in Figure 4.3. The contact pressure distribution cannot be reduced to a single normal force. Using only one vertical reaction force would keep the cube over the plane, but it could not prevent its rotation. Eventually, the cube will start spinning. A more complex reaction force model must be found in order to match the real pressure distribution at the contact area.

One simple solution is to place virtual contact primitives as spheres inside one of the objects, as in Figure 4.3. The spheres are adjusted to the corners of the cube, so the contact is approximated with those four points. When the cube is placed over the flat surface, each sphere provides a different normal force magnitude depending on their individual indentation with respect to the plane. Thus, a four point force set is obtained from them, which can be used to compute the final normal force and the torque that keeps the cube's orientation, since the reaction forces over the spheres approximate or least resemble the real pressure distribution that would actually exist.

As the object's geometry grows more complex, a higher number of spheres should be placed inside it, in order to approximate its surface with those tangent volumes [65]. Unfortunately, two big disadvantages show up as the number of spheres to be taken into account rises: an automated method for placing the spheres and guessing their size must be used in order to alleviate the cumbersome process of having to do it manually, and, in addition, the computational cost of collision testing for a high number of spheres per object can become unaffordable if a high degree of accuracy is requested.

The volumetric force model by [47] is more generic, and does not require any special coordinate configuration, nor any further preprocessing for placing additional auxiliary geometries. It is based on the elastic foundation model, which considers the contact region of the bodies as an elastic shell of known thickness and stiffness. The rest of the body —the material inside the shell— is considered fully rigid. The shape of the deformed foundation mattress is used for computing

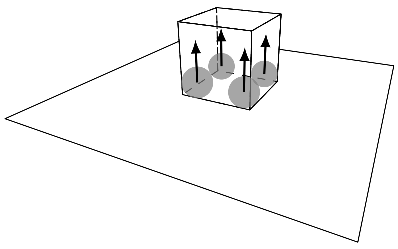


Figure 4.3: Placement of virtual collision primitives (spheres) for contact detection and force determination with conforming geometry.

the pressure distribution along the contact area between the bodies. From that area and the estimated deformation of the foundations, the normal reaction force and the rolling resistance torque can be computed. Furthermore, a decoupled friction model is available, so tangential and spinning friction can be also determined. The downside of the method is that it requires to compute at every time step the volumetric properties of the geometric intersection of the bodies. Geometric algorithms for finding the intersection volume between a pair of objects must be implemented for polygonal meshes, and later its mass center, volume and inertia tensor must be computed since the expressions of the model depend directly on those parameters.

For this model, the normal contact force $\mathbf{F}_n$ is computed as (4.6).

$$\mathbf{F}_n = KV(1 + a\mathbf{v}_{cn})\mathbf{n} \tag{4.6}$$

K is a stiffness constant computed from the elastic modulus of the foundation k_f and its depth h_f, $K = k_f/h_f$; a depends on the coefficient of restitution and the initial impact velocity. $\mathbf{v}_{cn}$ is the normal component of the relative velocity of one of the objects. Finally, V is the volume of the intersection between the bodies' surfaces.

The rolling resistance torque provided by the model, $\mathbf{T}_r$ (4.7), simulates the offset of the application of $\mathbf{F}_n$ from the centroid of the common contact surface.

$$\mathbf{T}_r = Ka\mathbf{J}_c\boldsymbol{\omega}_t \tag{4.7}$$

$\mathbf{J}_c$ is the inertia tensor of the interpenetration volume at its centroid, while $\boldsymbol{\omega}_t$ is the tangential component of the relative angular velocity between the objects, $\boldsymbol{\omega}$, that is contained in the contact surface.

The friction model provided by the model includes a tangential force $\mathbf{F}_t$ expression, (4.8): $\boldsymbol{\rho}_n$ is the vector going from the centroid of the volume to the centroid of the contact surface. For stiff bodies or bodies with similar stiffness, $\|\boldsymbol{\rho}_n\| \approx 0$ and thus, a simplified form (4.9) of the expression can be used.

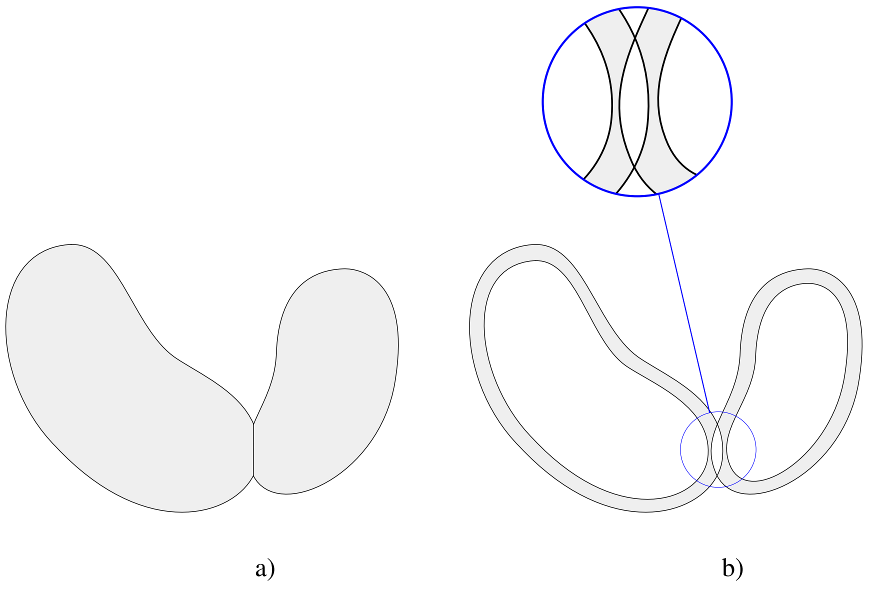


Figure 4.4: The elastic foundation model considers a real contact between solids a) as the mutual interpenetration of each one into the elastic shell of the other b). This model can deal with conforming geometries.

$$\mathbf{F}_t = \mu \mathbf{F}_n (\mathbf{v}_{ct} + \boldsymbol{\omega}_t \times \boldsymbol{\rho}_n) \tag{4.8}$$
$$\mathbf{F}_t \approx \mu \mathbf{F}_n \mathbf{v}_{ct} \tag{4.9}$$

The spinning torque model uses the friction coefficient μ and the normal component of the angular velocity $\boldsymbol{\omega}_n = \boldsymbol{\omega} - \boldsymbol{\omega}_t$, (4.7).

$$\mathbf{T}_s = \frac{\mu ||\mathbf{F}_n||}{V} \mathbf{J}_c \boldsymbol{\omega}_n \tag{4.10}$$

4.3. Geometry description

Usually, mechanical problems are solved without the need of specifying the exact geometrical shape of the bodies involved. Dynamic motion equations are expressed in terms of mass properties, since the mass distribution of a body can be reduced to a minimal set of parameters: mass, center of mass, and the inertia tensor.

When dealing with contacts, the definition of the surfaces is needed for computing certain properties. Many of those features can be computed before the simulation, therefore saving computation resources at run time. On the other hand, there is a hardware limit to the size of pre-computed data per geometry. It will be shown that there exist hardware penalizations when too big data structures are used, aside from memory capacity considerations. Property lists can grow so much that the program can incur into memory penalties when accessing the data. It could be faster, for example, to compute the normal vector of a surface than restoring it from memory, under some circumstances.

As a rule of thumb, properties that cannot be computed in a predefined amount of time should be stored from the beginning of the simulation, and the rest should be computed in real time, unless it is proven that the program can perform faster.

4.4. Far detection

Detecting contacts in a simulation is far from trivial. As the number of potentially colliding entities rises, the number of tests to be performed to check for collisions grows exponentially. Unlike simpler systems with few parts, as the system size increases, it is no longer possible to foresee specific contact configurations or to test any possible combination of objects for collision.

Far detection is the phase where it is determined which objects are potentially colliding in the frame of the entire simulation. As the objects in the simulation increase in number, naively checking for collisions between every possible couple of objects is not possible within a reasonable time frame. The computation time would grow exponentially. Clearly, some characteristics of the scene can be exploited for speeding up the process. The performance of any particular method depends on the nature of the system and its environment. There are better-suited techniques for detecting collisions between moving objects and fixed entities, and other procedures that deal specifically with moving objects only.

Space partitioning techniques [9] are heavily used to achieve the goal of computing the list of colliding objects so their reaction and friction forces can be calculated afterwards. In a first stage, the whole space for the simulation is partitioned in order to help to quickly discard non-colliding objects.

Spatial segmentation techniques are based on the cell division of the space covered by the static, immutable scenery. If a moving object is detected to be inside a cell, it can be only checked for collision against the objects inside that cell, thus discarding all the objects within the other cells.

Still, just dividing the space into cells it is not enough to avoid the exponential cost of having to perform collision checks between an object and all the cells for the first time. The size of the cells is also relevant: if they are too big, there will be many objects to check inside them, defeating the purpose of the space division; if they are too small, there will exist a cell-finding overhead, since they will have to be a large amount to cover the space. Either way, the cost of checking a group of bodies against the rest of the objects is exponential, $O(n^2)$, since it requires checking each one against the rest, (4.11).

$$C(n,2) = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} \tag{4.11}$$

Recursive or tree-based methods are almost the best suited solution, as they can vastly reduce the number of checks. They consist on applying a subdivision to each cell several times, until an finalizing condition is met. Usually it is expressed in terms of the number of objects remaining in the smallest cell. These division schemes are stored as trees or graphs, given that each cell can spawn several children. The complexity is reduced to $O(n\log(n))$. For a binary tree, the feature set is halved at each level, resulting in a total depth of $\log_2(n)$ steps. The depth of the tree is usually related to the maximum number of necessary checks to perform.

4.4.1. Octrees

The *octree* classification model can segment the space into eight regular cells, which in turn, will be divided eight times as well [37]. The subdivision of each cell box is made by splitting it along the axes of the global frame —hence the eight parts. Each cell is associated with the set of scenery features that can only be stored into its boundary as a minimum, and with its eight spawned divisions: if an object does not fit into any of the child boxes, it remains into the parent cell; otherwise, it is assigned to any of the children boxes.

In order to find objects into the octree that could potentially collide with an object, a tree traversal is started from the root cell. At each level, cells where the object is contained are checked, while the rest is discarded. The process continues for the children cells, until a leaf of the tree is found.

Octrees are specially well-suited when the objects they contain are distributed homogeneously over the space. Its partitioning algorithm is completely regular, so a cell will occupy the same space independendly from the number of objects it holds. If there exists many empty areas, some cells will be wasted covering those empty regions.

4.4.2. BSP trees

Another feature-classification models include the *k-dimensional* trees or *k-d trees* [8] and the *Binary Space Partitioning* trees, *BSP trees* [39]. Instead of segmenting the space against an homogeneous grid partitioning, those models split

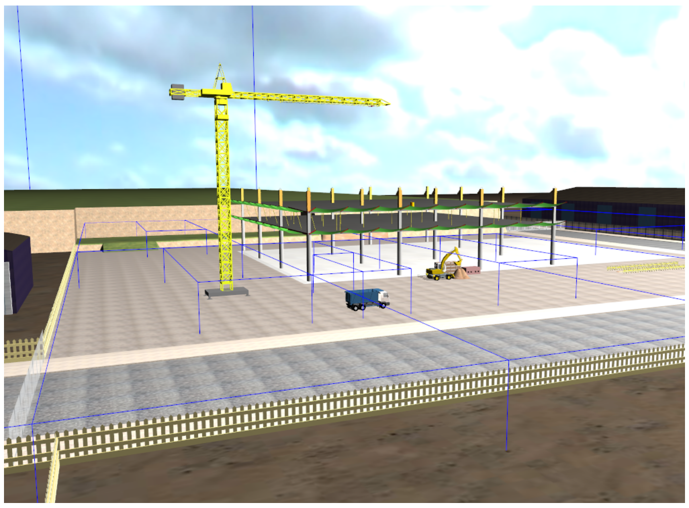


Figure 4.5: Octree spatial subdivision in a simulator scenery. Note how the divisions are finer at zones with a higher level of detail, and that they are aligned to the subdivision planes.

the space at each level into two regions. Objects placed at each side of the partitioning plane are assigned to their corresponding region. Those regions are further split independently, also deriving into a tree shaped structure.

In the case of the *k-d trees*, the splitting plane can be parallel to any of the orthogonal planes, $x = 0$, $y = 0$ or $z = 0$. At each level, it is chosen the one that halves the maximum dimension of the space. Meanwhile, *BSP trees* are more flexible, and admit any plane for the division operation.

One advantage of those models over the octree is that they can handle fairly well scenery data sets whose feature density is non-homogeneous. BSP trees and k-d trees will only partition against existing geometry, almost eliminating the chances of having empty subdivisions. In addition, a given data set has not an unique BSP of k-d tree sorting. As shown, at every subdivision step, a dividing plane must be chosen for computing the next split. That choice will directly influence the balancing and depth of the resulting tree. An unbalanced tree can perform almost as poorly as having no sorting at all. Several strategies for choosing the splitting feature can be used. Usually each split is determined by choosing the most-centered feature of the available space, or the one that would have as many features present on both sides of the created half-space.

A specific advantage of the BSP tree structure is that, at every node, it defines two half-spaces that can be used to match the outside and the inside regions whose common frontier is the splitting polygon of the node. This property is very useful for storing the enclosed volume of a manifold body geometry, as

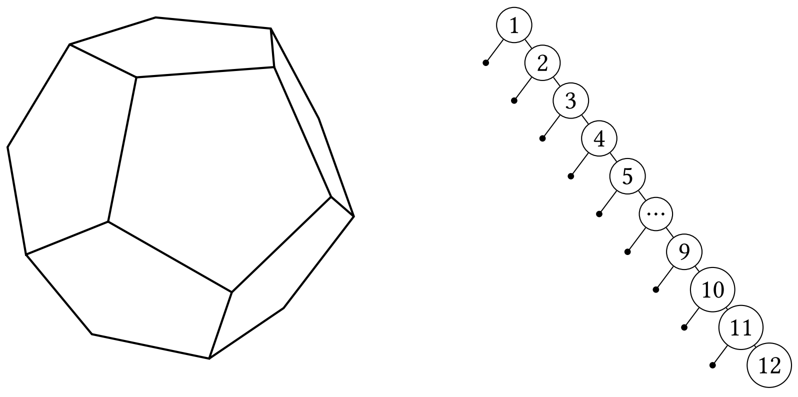


Figure 4.6: Selecting the facets of a convex polyhedron as the splitting planes for the BSP leads to completely unbalanced trees.

Boolean operations between geometries can be built upon this technique [104, 82].

The downside of BSP trees and k-d trees is that the splitting process can lead to the subdivision of any feature that could be lying on both sides of the bisecting plane. It must be divided and its fragments assigned to each of the sides. As a net effect, this increases the problem size, since the data set would grow because of the classification process.

An easy splitting policy is to use the planes defined by the polyhedron's facets as dividing planes. However, if the polyhedron is convex, a fundamental flaw exists: the resulting tree is completely unbalanced, since in a convex polyhedron, the whole volume is always contained within the interior side of every facet. Thus, all the remaining facets to classify will always lie at one side of the plane, and no division will be made (Figure 4.6). This results in a linear tree. That problem can be solved by using different splitting planes than the facets, although the process will incur into facet fragmentation.

4.4.3. Object against static tree tests

At simulation time, checks are made between the objects to test for collisions and the preprocessed scenery. Usually the objects are enclosed into simpler, primitive bounding volumes that can speed up checks against every node belonging to the tree. Bounding volumes such as boxes or spheres have simple definitions and lead to small and quick collision tests. Those tests will be described in §4.5. The effect of using enclosing volumes can result into a small performance penalty, as they cover more space than the actual geometry, thus potentially triggering false positives. Nevertheless, when a fine-grained level check is performed, the false positives are discarded.

Typically, the bounding volume enclosing the geometry is tested for collision with the first node of the tree. If the check is positive, the bounding volume is then checked against the child nodes belonging to the portion of space that lead to the positive result. For the Octree case, the bounding volume would be checked against any of the eight possible children that the volume is touching. For BSPs, the volume would be tested against the children of either side of the plane where the bounding volume were located.

During the tree traversal, features belonging to every tree node traversed are stored in a list, in order to perform an eventual finer collision test against the real geometry.

4.4.4. Collision tests for moving objects

Collision detection between moving objects is more involved than the methods seen before. Since the placement of any object can vary at each time instant, it is not possible to rely on space partitioning techniques: rebuild or update a partitioning tree at each instant of time would ruin the performance of the simulator. Nevertheless, some properties of the scene can still be used in order not to having to fall back to perform all the possible collision combinations.

Usually, moving objects in a simulator scene are parts that belong to bigger mechanism assemblies. The position of each of those parts is fixed or it does not vary too much within the mechanism frame. Hence, it is a good strategy to enclose all the mechanism together into a single bounding box. Therefore, if it is needed to check an object for collision against any part in the mechanism, it could be checked first against the mechanism's bounding volume. If the test fails, the individual checks against each part is avoided.

In the same spirit, a mechanism could be divided into sub-mechanisms if it is expected for a group of parts not to have large relative displacements. In Figure 4.8, an example layout of the bounding volumes for a excavator machine is presented. Since the cabin and the wheels are likely to be always grouped together, it makes sense to create a bounding volume that encloses all those parts. However, the rest of the parts (the boom, the stick and the bucket) can have a great variation in position during the machine operation, so they are assigned their own individual volumes. Finally, a global volume is created to enclose the group and the individual parts.

This classification defines a hierarchy in the assembly of the parts of the mechanism. Like the static cases, the hierarchy classification can help to discard a high number of collision tests. The downside lies in that the classification has to be carried by the simulator developer, since it cannot be automatically computed as the former cases. Knowledge of the possible configurations for the

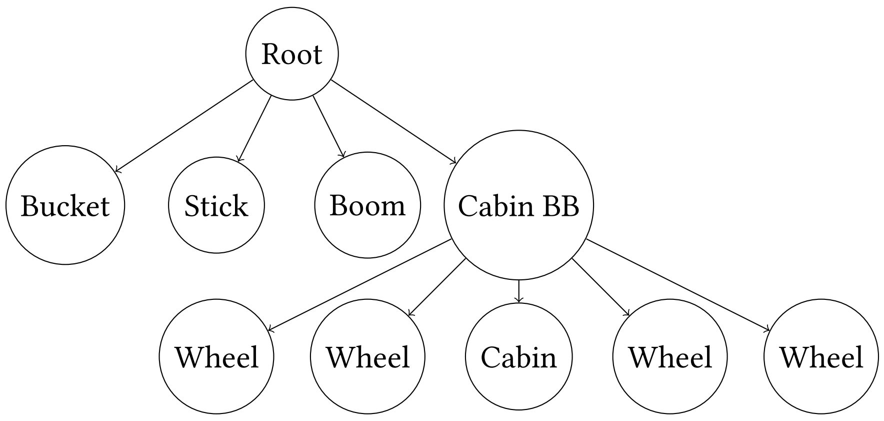


Figure 4.7: DAG for the excavator assembly.

machine is required in order to find the best partitioning for the mechanism's elements.

Each hierarchy relationship among parts can be expressed as a *Directed Acyclic Graph* (DAG). The DAG is a set of interconnected nodes (the bounding volumes) which can only traversed from the root nodes to the children. For the hierarchy to be effective rejecting additional collision tests, a node's bounding volume has to enclose the bounding volumes of its children. As the parts change their position in the simulation, the bounding boxes of the DAG have to be updated.

Bounding spheres are the most appropriate choice for the bounding volumes of mobile parts, since they are easily updated when the children of a certain node changes its position. The bounding sphere of center $\mathbf{c}_n$ and radius r_n that encloses two given bounding spheres whose centers are $\mathbf{c}_1$ and $\mathbf{c}_2$ and radius r_1 and r_2 can be calculated as

$$d = \|\mathbf{c}_1 - \mathbf{c}_2\| \tag{4.12}$$

$$r_n = \frac{d + r_1 + r_2}{2} \tag{4.13}$$

$$\mathbf{c}_n = \frac{\mathbf{c}_1(d + r_1 - r_2) + \mathbf{c}_2(d - r_1 + r_2)}{d} \tag{4.14}$$

If the distance d is much larger than r_1 and r_2, the bounding sphere fitting will be worse, and the collision test false positives will increase.

The procedure for colliding two DAG hierarchies is:

1. Pick two DAG hierarchies and test their root bounding volumes. If the test is positive, proceed to next step.

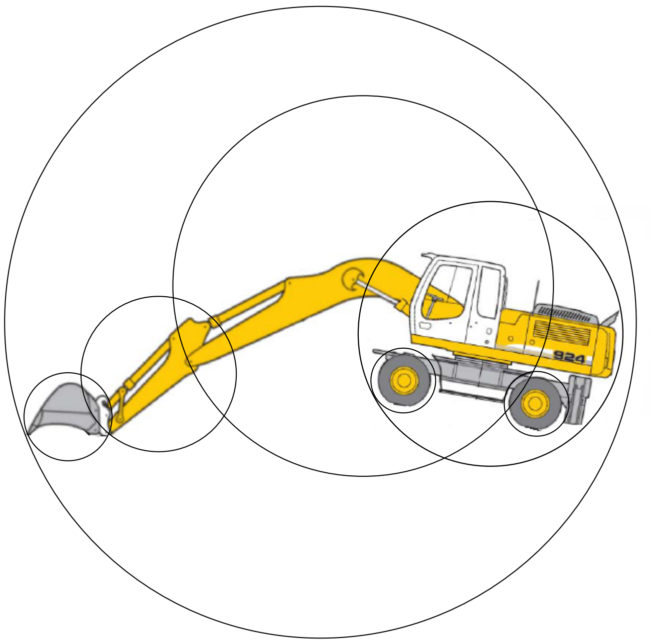


Figure 4.8: A bounding sphere hierarchy is shown for the parts of the machine. The parent bounding primitive encloses a volume much larger than the one that the machine actually occupies given its configuration.

2. Pick the children of each node of the first object and the second object. Perform all the possible test combinations between them.
3. Discard any node that has not been found in collision at least once.
4. For the rest of the nodes, repeat step 2.

4.5. Primitive-Primitive detection

The term "collision *primitive*" is used in this document for referring to very simple shapes used for collision detection purposes. Spheres, rectangles, cylinders... are examples of collision primitives that can be used to enclose more complex geometries and get an initial estimation of the contact state.

Primitive-primitive collision detection can be regarded as the simplest collision detection between entities, since often their shape can be written as well-known, simple, analytic expressions. The evaluation of those expressions allows to compute the minimum distance between pairs of primitives. Unlike other geometry approximations, the accuracy of the checks do not depend on any resolution parameter.

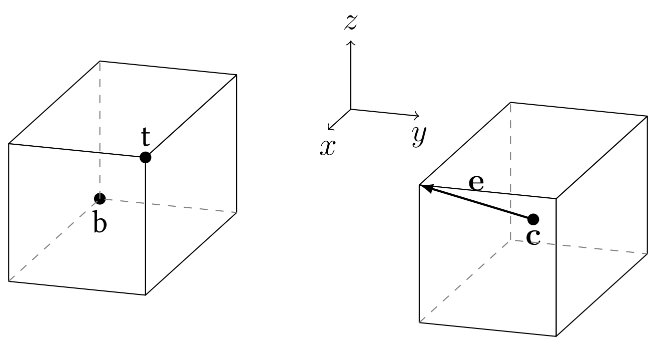


Figure 4.9: Left: AABB definition by extreme points. Right: AABB definition by center and maximum extents.

This simplicity comes at the cost of reduced flexibility when a primitive is used as the bounding volume of an object. Usually the space covered by the primitive encloses much more volume than it is contained by the object, which often leads to false positives. This is usually solved by using a set of smaller primitives in order to define the bounding volume as the union of all of them. It still can be difficult to fit that set to the actual shape of the object without having to use a big number of primitives, which would decrease performance as a result of the increasing number of tests.

4.5.1. Axis-Aligned Bounding Boxes

A very useful primitive for collision tests is the *Axis-Aligned Bounding Box*, AABB, which can be used to enclose a variable number of features, from sets of points to facet *soups*. A big advantage of the AABB is the reduced number of parameters needed for its parametrization. The edges and facets of an AABB are always aligned to the global coordinate frame, and thus it is just commonly defined by a pair of spatial points. There exist two usual representations (Figure 4.9):

- Center and extension: defined by the position of the center of the box $\mathbf{c} = (c_x, c_y, c_z)$ and a vector holding its extension, $\mathbf{e} = (e_x, e_y, e_z)$ over each of the axes.
- Extreme points: defined by two opposite vertices of the box, $\mathbf{t} = (t_x, t_y, t_z)$ and $\mathbf{b} = (b_x, b_y, b_z)$.

Both definitions are equivalent, and it is a matter of convenience the use of one of them instead of the other, as demonstrated in Equation (4.15).

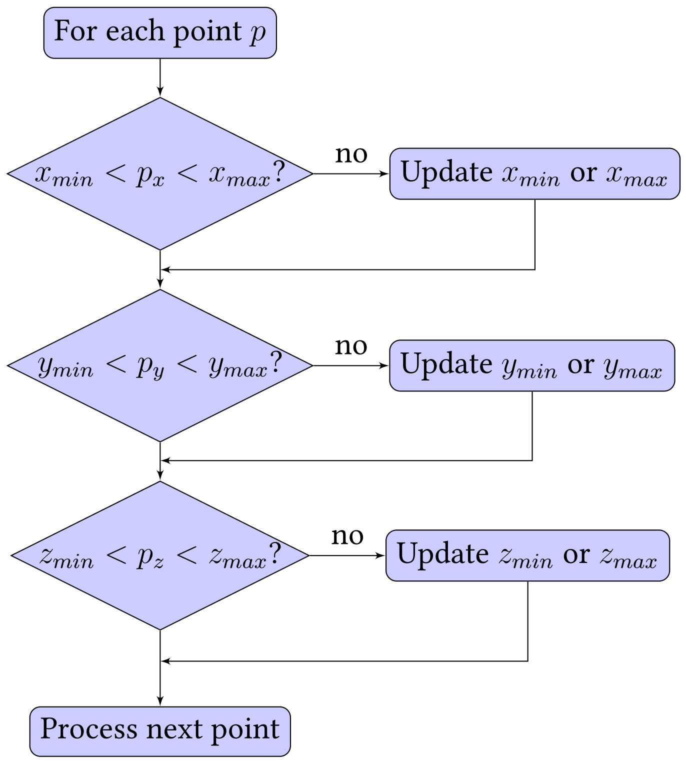

For each point p
$x_{min} < p_x < x_{max}$?
no
Update x_{min} or x_{max}
$y_{min} < p_y < y_{max}$?
no
Update y_{min} or y_{max}
$z_{min} < p_z < z_{max}$?
no
Update z_{min} or z_{max}
Process next point

Figure 4.10: AABB creation procedure.

$$\mathbf{c} = \frac{\mathbf{t}+\mathbf{b}}{2} \tag{4.15}$$

$$\mathbf{e} = \mathrm{abs}\left(\frac{\mathbf{t}-\mathbf{b}}{2}\right) \tag{4.16}$$

Another beneficial property of AABBs is that their making is a simple procedure. By finding the extreme coordinates of each entity over the global frames' axes, points $\mathbf{t}$ and $\mathbf{b}$ can be computed, and optionally later transformed into the form $[\mathbf{c}, \mathbf{e}]$ as shown in equation(4.15).

For entities as points, edges or facets, the vertices defining them are the ones used in the computation of the extremes. From a zero-sized AABB, each time that one of the points of the list is found to lie outside, the box is enlarged in order to fit it. Note that the box is always enlarged, never shrunk: the box is not set to the coordinates of outer points, it is only grown over the axis needed to fit them. Figure 4.10 shows a flowchart of the procedure.

AABBs can be created from any type of geometric entity, as long as a method for determining their maximum extents over each of the axis is provided. They can be used to enclose point clouds, segments, polygons, whole meshes or any kind of shape defined by an analytic expression.

The collision test between a pair of AABBs is trivial: two AABBs are in contact if, for the three coordinate axes, the ranges for each box over each axis are coincident. This is equivalent to fulfill the three statements in (4.17).

$$\begin{aligned} \left[x_{min}^A, x_{max}^A\right] \cap \left[x_{min}^B, x_{max}^B\right] \neq \emptyset \\ \left[y_{min}^A, y_{max}^A\right] \cap \left[y_{min}^B, y_{max}^B\right] \neq \emptyset \\ \left[z_{min}^A, z_{max}^A\right] \cap \left[z_{min}^B, z_{max}^B\right] \neq \emptyset \end{aligned} \tag{4.17}$$

4.5.2. Box-Sphere

This test is very useful when using boxes and spheres as bounding volumes for more complex geometrical objects [9]. If those contents are tightly fitted to them, minimizing unused space, the test hugely accelerates the task of discarding non-colliding geometries. The test is computationally cheap to perform because of the simple definitions of those bounding volumes.

The test has to steps: first, computing the nearest point from the box to the sphere; then, checking if the distance from that point is greater than the radius of the sphere. If that is the case, the sphere is found not to be touching the box.

This test assumes that the box is aligned to the global frame of reference, that is, the box is an AABB; if not, the sphere center has to be transformed into the local coordinate frame of the box. The box is aligned to its coordinate frame, thus meeting the definition of an AABB in that frame. If the frame of the box is defined by the unit, orthogonal vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$, any point in the global space can be expressed in terms of the local frame as shown in equations (4.18) - (4.20).

$$\mathbf{T} = \begin{pmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} \end{pmatrix} ; \mathbf{T}^{-1} = \mathbf{T}^{\mathrm{T}} = \begin{pmatrix} \mathbf{u}^{\mathrm{T}} \\ \mathbf{v}^{\mathrm{T}} \\ \mathbf{w}^{\mathrm{T}} \end{pmatrix} \tag{4.18}$$

$$\mathbf{r} = \mathbf{r}_0 + \mathbf{T}\overline{\mathbf{r}} \ \Rightarrow \overline{\mathbf{r}} = \mathbf{T}^{-1}(\mathbf{r} - \mathbf{r}_0) \tag{4.19}$$

$$\overline{\mathbf{r}} = \begin{pmatrix} \mathbf{u}^{\mathrm{T}}(\mathbf{r} - \mathbf{r}_0) \\ \mathbf{v}^{\mathrm{T}}(\mathbf{r} - \mathbf{r}_0) \\ \mathbf{w}^{\mathrm{T}}(\mathbf{r} - \mathbf{r}_0) \end{pmatrix} \tag{4.20}$$

Let $\mathbf{n}$ be the nearest point of the box to the center of the sphere, $\mathbf{c}$. If $\mathbf{n}$ is found to be outside of the sphere, then both volumes are not in contact. It is very convenient that both primitives are expressed in a coordinate frame where the box is aligned to its axes, since it simplifies the computation of $\mathbf{n}$. In that frame, the box is an AABB.

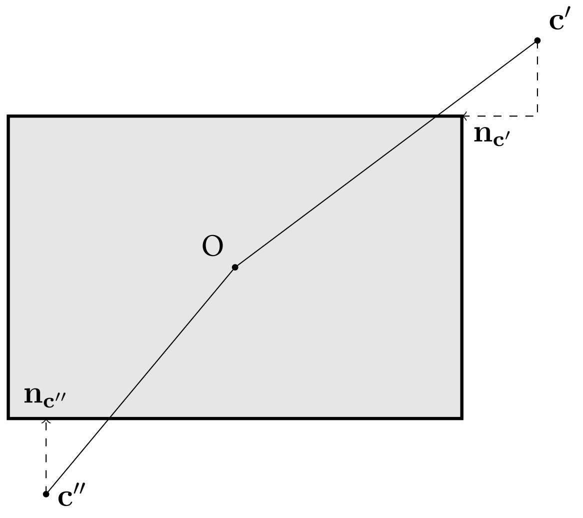


Figure 4.11: Nearest point from the box to the center: each component is clamped to the extents of the box.

The point $\mathbf{n}$ is located at the surface of the box if the center $\mathbf{c}$ lies out of its volume. $\mathbf{n}$ can be computed projecting $\mathbf{c}$ over the faces of the box. Since all the faces are aligned, the projection is greatly simplified.

Let $\mathbf{v} = \mathbf{c} - \mathbf{p}$ be the vector joining the centers of the box and the sphere. Each component of $\mathbf{v}$, v_i, is clamped to the extents of the box in that direction, e_i. The process is defined by equation (4.21): if v_i lies in the range $(-e_i, e_i)$, it is left unmodified. On the other hand, if $v_i > e_i$ or $v_i < -e_i$, its value is set to e_i or $-e_i$, respectively.

Figure 4.11 illustrates the procedure for finding the nearest points in the box with respect two different points, $\mathbf{c}'$ and $\mathbf{c}''$. The horizontal component of $\mathbf{c}''$ is contained into the horizontal range of the box, so it is left unmodified. However, its vertical component lies outside of the box, so it is clamped to the bottom plane, resulting in the nearest point $\mathbf{n}_{\mathbf{c}''}$ for the box. In the case for $\mathbf{c}'$, both of its components lie outside of the horizontal and vertical range, and thus they are clamped in successive steps. The result is $\mathbf{n}_{\mathbf{c}'}$.

$$\max(-e_i, v_i) \leq n_i \leq \min(e_i, v_i) \tag{4.21}$$

The sphere is touching the box if the distance from $\mathbf{n}$ to $\mathbf{c}$ is smaller than the radius of the former:

$$(\mathbf{n} - \mathbf{c})^{\mathrm{T}}(\mathbf{n} - \mathbf{c}) \leq r^2 \tag{4.22}$$

For force computation purposes, the maximum indentation can be computed by means of the minimum distance between the sphere and the box. If the sphere is touching the box only on one side, the minimum distance will be the length of the perpendicular segment to the plane that joins the center of the sphere with the plane. The normal vector determining the force direction is the normal vector of the plane. If the sphere touches two of the box faces, the minimum distance is the segment between the center of the sphere and both faces' common edge. The segment is perpendicular to the edge. The normal will have the direction of the vector composed by the point and the intersection between the segment and the edge. Finally, if the sphere touches three of the box planes, the minimum distance point will be the distance to the shared corner of the three faces, and the force direction will be described by the those two points.

4.5.3. Box-Box collision

As seen, most of the time the convenience of using AABBs as bounding volumes lies in the simplicity of its definition, based on the global coordinate frame. When an AABB is tested against any other primitive, it is very frequent to be able to transform it into the coordinate frame of the AABB, therefore simplifying the computations. However, there exist cases where it is not advisable to use an AABB as a bounding volume for an object: it could be that the global frame does not determine the best fitting box for the object; in that case, the volume of the AABB would be much larger than the object, thus yielding a high rate of false test positives. For objects that move and rotate, their AABB must be computed at each simulation step, and their fitting will vary greatly depending on its orientation.

To overcome those disadvantages, a bounding box fixed to the local frame of each object can be used. This kind of volume is referred to as a *Oriented Bounding Box*, OBB. The general case for arbitrarily oriented bounding boxes is more complex than the AABB case, but nevertheless, is still a fast approximation for the initial stages of the collision tests.

In the planar case, when testing two rectangles for collision, it is enough to find a vertex from any of them to be inside the other shape to determine that both are touching. This is done searching for a vertex point lying at the back inner side of all the sides from the other shape. Unfortunately, there is no three-dimensional equivalent for this algorithm: as described in [33], there are special orientations for which the boxes can be touching, yet none of the vertices of any of the boxes is placed inside the other.

An alternative is to use the *Separating Axis Theorem* [52], which states that two boundary volumes are assured not to be in contact if a plane is found where their projections over the plane do not overlap. When using planar faceted bounding volumes, the set of directions to try is determined by the planes of

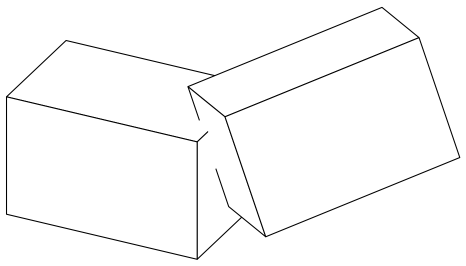


Figure 4.12: For three-dimensional boxes, there could exist a collision even if all of the vertices of one box are not enclosed inside the other.

the facets, and the ones defined by cross products between their edges. In the box-box case, there are 15 directions to try, being 6 of them the corresponding to the 3 axes for each box, and the remaining 9 the possible cross-products between them (Table 4.1).

Type	Expression	Quantity
Axes of box A	$\mathbf{u}_i^A$	3
Axes of box B	$\mathbf{u}_i^B$	3
Cross product of axes A and B	$\mathbf{u}_i^A \times \mathbf{u}_i^B$	9

Table 4.1: Separating axes for two boxes.

For each direction $\mathbf{d}_i$, a check is made in order to know if the length of the segment from one box center to the other is greater than the sum of the extensions of the boxes over that line:

$$\|(\mathbf{c}^A - \mathbf{c}^B)^{\mathrm{T}}\mathbf{d}_i\| > \|(\mathbf{p}^A + \mathbf{p}^B)^{\mathrm{T}}\mathbf{d}_i\| \tag{4.23}$$

where $\mathbf{c}^A$ and $\mathbf{c}^B$ are the centers of the boxes, and $\mathbf{p}^A$ and $\mathbf{p}^B$ are the nearest points from box A and B to $\mathbf{c}^B$ and $\mathbf{c}^A$, respectively. For computing the nearest points, the same algorithm described in §4.5.2 is used. Although contrived, the SAT method is faster than comparing each edge in a box against all the facets of the other.

4.6. Primitive-Mesh detection

Primitive-Mesh detection is the last step before having to resort to full mesh-mesh contact detection. Not only it usually leads to faster tests, but in addition it can provide better contact parameter accuracy, polygonal mesh surfaces are approximated.

In this document, mesh surfaces are considered as a set of flat, polygonal faces that enclose a volume. It is not usually needed for them to define a manifold or to be convex, although some of the algorithms require it. The contact tests against each primitive are performed in a facet by facet basis. Nevertheless, usually the set of facets tested for collision is the subset that it is found to be potentially colliding with the primitive. Testing against a reduced set of facets improves the overall performance of the methods.

4.6.1. OBB trees

There exists an analogy between the far collision detection and the detailed collision detection between a polygon mesh and any other geometric entity. Approximating surfaces by polygonal meshes is advantageous in terms of algorithm simplification and generalization: any surface can be expressed as a set of polygonal facets, and this representation can be obtained automatically from CAD programs or any other type of designing software. However, as the polygon count is higher, the performance cost of testing each polygon in the mesh against other primitives or polygons becomes unaffordable.

As described in the far detection case, polygon partitioning methods are needed in order to have to do the fewest possible number of tests. For polygon meshes, BSP trees, OBB trees and k-d trees are used. Usually the trees partition the polygons or the space occupied by the polygons into smaller bins at every level. Since the main interest when testing two objects for collision is to find which the exact contact points are, this detail usually requires to reach individual polygons at the leaves level. Therefore, it can be known which specific polygons in the mesh are potentially intersecting with the other object. Consequently, these mesh structures have a higher node/polygon ratio, usually being 1:1. This high node count incurs into a bigger memory fingerprint.

The consequences of the higher requirements in node number and memory are that methods which lead to shallower trees and whose nodes have a smaller memory requirements will perform better. The shallower the tree, the faster it is discarding non-colliding polygons, since it is easier to reach the leaves of the tree in fewer steps. This is directly influenced by the splitting policy followed when the trees are built. This happens when building BSP and OBB trees. Possible policies for BSP splitting were discussed in past sections.

OBB trees [53] could be regarded as a subset of BSP trees where the space division at each node is aligned with the Cartesian planes of the local frame of the object. In that frame, the boxes of the tree are AABBs. A particular case exists when the root node of the OBB is aligned with respect to the global frame: in that case, it is called an AABB tree, since all its nodes are aligned to the global frame, and therefore they are AABBs.

The space volume at every node is contained into an AABB; at the following level, they are subsequently split into two new AABBs. The subdivision process for a branch ends when its node, the AABB, holds only one polygon. This node is called a *leaf*.

The subdivision strategy for an AABB node is not unique; different policies lead to trees with varying quality levels, in terms of performance. Dividing each AABB node into two equal-sized parts is equivalent to create an octree. Dividing the nodes in order to have similar polygon densities leads to better balanced trees. Two decisions must be taken: which plane should be used for the division, and at which point into the parent AABB that plane should be placed. To solve both questions, the following procedure is applied:

1. The splitting point is computed as the barycenter of all the polygon's vertices:

$$\mathbf{p}_s = \frac{\sum_{i=1}^{n_p} \sum_{j=1}^{3} \mathbf{v}_j^i}{n} \tag{4.24}$$

2. Next, calculate the standard deviation for each of the axis:

$$\boldsymbol{\sigma}^2 = \frac{1}{n_p - 1} \sum_{i=1}^{n_p} (\sum_{j=1}^{3} \mathbf{v}_j^i - \mathbf{p}_s)^2 \tag{4.25}$$

   The best candidate axis for splitting the AABB is the one with the biggest $\boldsymbol{\sigma} = (\sigma_x, \sigma_y, \sigma_z)$ dispersion component.

A polygon is assigned to one of the half-spaces defined by the splitting plane. However, there are polygons whose vertices lie on both sides of that plane. The OBB tree avoids fragmenting polygons by assigning them to one of the sides. A good criteria is to classify a polygon against the plane depending on which side its barycenter lies in (Figure 4.13).

Since each AABB node can contain polygons that lie partially in both half-spaces, there exist some overlap between sibling nodes. Nevertheless, the policy of not fragmenting polygons avoids the growth of the tree, as it happens with BSP trees.

OBB tree-bounding volume collision test

A collision test between an OBB tree and a bounding volume — such as a sphere or a box — returns the facets of the tree that are touching or inside the

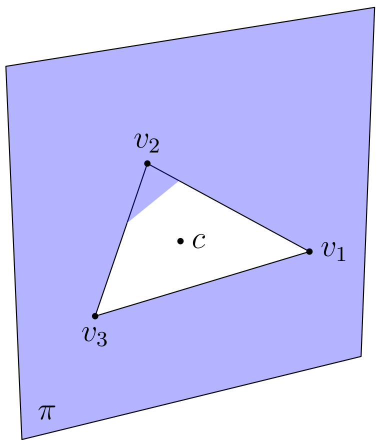


Figure 4.13: To avoid fragmenting polygons that have vertices on both sides of the splitting plane π, they are assigned to the side where its centroid c lies.

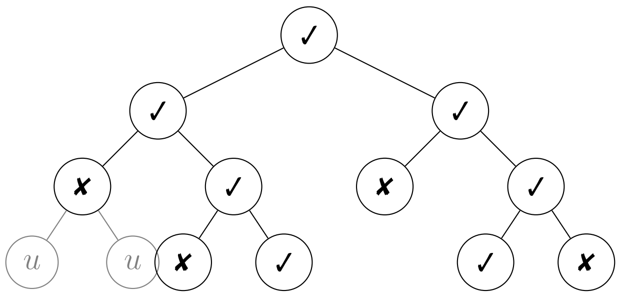


Figure 4.14: AABB-tree traversal from the root node. Failed checks (✗) allow to discard the rest of the sub-tree, therefore reducing the number of checks to perform. Untested nodes are signaled as u.

volume. The test is done checking the AABB node of the tree against the bounding volume. The algorithm descends the tree only for the nodes that are found to be in contact with the volume. The rest of the nodes and their offspring are thus discarded, see Figure 4.14.

At each tree level, the bounding box can be found to be in contact with both, one or none of the nodes. If both nodes are contacting the volume, the tree can be descended for the first node and their children, and then for the second node and their children. Recursive functions are well fitted for this scheme.

The descent ends when a leaf node is reached, if the check for that node is positive, the facet it holds is added to the list of colliding polygons. That list will be used to perform the contact parameter computation against the object inside the bounding volume. The details for the individual polygon-object tests are described in the following sections.

OBB tree-OBB tree collision test

Collision tests between two OBB trees are more complex since both trees have to be traversed in parallel. After a successful test between the root nodes of both trees, the child nodes of a tree must be checked against the children of the other tree. There are 4 tests to be carried between the nodes, shown in (4.26). n_i^T is the i^{th} node of the T tree.

$$\begin{array}{c} (n_1^A, n_1^B) \\ (n_1^A, n_2^B) \\ (n_2^A, n_1^B) \\ (n_2^A, n_2^B) \end{array} \tag{4.26}$$

Again, a recursive descent is well suited for traversing the trees. However, if at each level the traversals are performed in the same order, e.g. node 1 of the *A* tree is always traversed first if its test is positive, the traversal will compare the deeper nodes of *A* against the higher nodes of *B*. This is counterproductive because there is a high probability that the smaller child AABBs of *A* are always contained into the bigger box of *B*, defeating the purpose of the AABB-tree segmentation, since the whole sub-tree of *A* has to be compared against that *B* node.

A good strategy for descending both trees in an evenly manner is to traverse the biggest AABB. When a overlapping pair of AABB nodes is found, the next check will test the smaller AABB node against the two children of the other tree node. The AABB boxes being compared will be of a similar size, and therefore the checks will be more accurate.

As in the previous section, the traversal over the branches of both trees end when there are found two overlapping leaf nodes. The pair of polygons contained into each leaf is stored in a list for the fine-grained collision test at a later stage.

4.6.2. Mesh-Sphere

This type of contact is the simplest of any primitive-mesh test. Once all the potentially colliding polygons are known, the parameters needed for the contact force computation are retrieved from simple formulae. At a later stage, the list of colliding polygons can be interpreted as desired, depending on the used force model. Usually force models are defined in terms of indentation and the surface normal at the contact point. Extracting those parameters from the list of polygons could be made in several, different ways.

Here is to be described the case for a sphere versus triangle mesh collision test. A generalization for considering any polygon mesh will be described later.

In the case that the number of polygons in the mesh is large, it makes sense to use first a far-detection algorithm in order to work only with the subset of potentially colliding faces instead of the whole mesh. Afterwards, the list of facets will be tested against the sphere primitive. Each polygon is tested in two stages. First, it is computed whether the sphere touches the plane where the facet contained. If the check is successful, it is checked if that intersection is contained inside the polygon.

Face collision detection

The plane containing a polygon can be computed from a perpendicular vector and one of its points. For a triangle, the normal vector and the point can be calculated from its vertices, as shown in the equation (4.103). This procedure is very sensible to the quality of the triangle, as will be seen in §4.7. If the polygons of the mesh are degenerated, a least squares fitting procedure can be applied, as shown in §4.7.4.

Once the plane π of the polygon is computed, the minimum distance to the sphere δ is calculated. The plane (4.27) is expressed in terms of its normal $\mathbf{n}$ and its distance to the origin, d. For all the points in the plane, $\mathbf{x}$, (4.27) evaluates to zero. For any other point, the plane equation π evaluates to the minimum distance from the point to the plane.

$$\pi : \; \mathbf{n}^{\mathrm{T}}\mathbf{x} + d = 0 \tag{4.27}$$

$$R_{sphere} - \delta = \mathbf{n}^{\mathrm{T}}\mathbf{p}_{center} + d \tag{4.28}$$

δ is the minimum distance from the sphere whose center is located at $\mathbf{p}_{center}$ (see Figure 3.1). There is a contact between the sphere and the plane if the indentation δ is greater than zero, $\delta > 0$. The contact point $\mathbf{p}_{contact}$ is computed as

$$\mathbf{p}_{contact} = \mathbf{p}_{center} - (R_{sphere} - \delta)\mathbf{n} \tag{4.29}$$

However, the area of the intersection of the sphere and the plane could still lie outside of the polygon's boundary. If the projection of the center point over the plane π lies inside the polygon, it is assured that both are in contact.

A quick test for determining if a point is found inside of a polygon is to compute the *barycentric coordinates* of the point within the polygon. The barycentric coordinates form a coordinate system based on the location of the vertices of the polygons. The barycentric coordinates λ_i of a point $\mathbf{P}$ are related by the expressions (4.30).

$$\sum_{i=1}^{n} \lambda_i = 1 \qquad \mathbf{P} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \tag{4.30}$$

where n is the number of vertices of the polygon. The barycentric coordinates of a point with respect to a triangle are three, and they always sum 1 by definition. A point lies inside a polygon if its barycentric coordinates λ_i meet the requirement (4.31).

$$0 \leq \lambda_i \leq 1 \tag{4.31}$$

As remarked, if the requirements (4.31) are fulfilled, the projected center of the sphere is inside the triangle, and there exists a contact. It is defined by the distance δ and the contact normal $\mathbf{n}$. Thus, the barycentric coordinates of a point $\mathbf{P}$ with respect to a triangle defined by three vertices $\mathbf{v}_i$ are computed solving the linear equation system

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{pmatrix} \boldsymbol{\lambda} = \mathbf{P} \tag{4.32}$$

Only two of those equations in the system are independent, so the other can be removed from the system:

$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ 1 - \lambda_1 - \lambda_2 \end{pmatrix} = \mathbf{P} \tag{4.33}$$

$$\begin{pmatrix} \mathbf{v}_1 - \mathbf{v}_3 & \mathbf{v}_2 - \mathbf{v}_3 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \mathbf{P} - \mathbf{v}_3 \tag{4.34}$$

Edge collision detection

Nevertheless, if the requirements (4.31) are not fulfilled, the sphere could be still touching the polygon: the sphere could be pierced by one of its edges. If the sphere touches the plane but its center's projection does not lie into the polygon, each of its edges must be tested against the sphere.

Each edge of the polygon is defined by a pair of vertices, $\mathbf{e}_i = \mathbf{v}_j - \mathbf{v}_k$. The nearest point from the edge to the sphere is computed by the intersection between the edge and a perpendicular segment passing through $\mathbf{p}_{center}$. In the

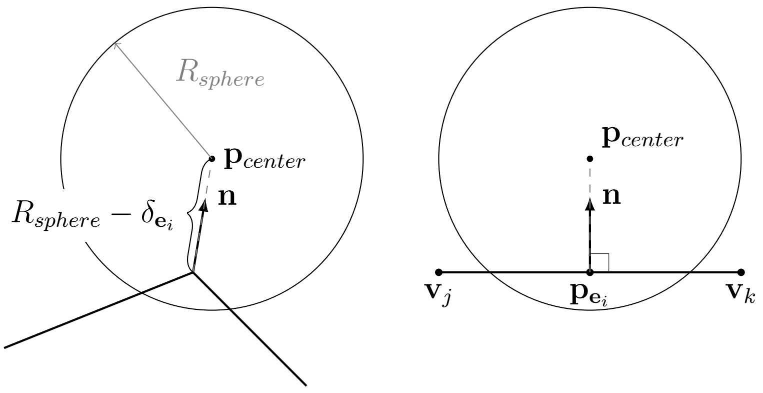


Figure 4.15: Frontal and lateral views for an edge contact.

same spirit as the face tests, the point $\mathbf{p}_{center}$ is projected into the edge segment. This point, $\mathbf{c}_{\mathbf{e}_i}$ is checked for its distance with respect to $\mathbf{p}_{center}$, and for its location into the edge (4.35). If $\delta_{\mathbf{e}_i} > 0$, the sphere is potentially colliding with the edge.

$$R_{sphere} - \delta_{\mathbf{e}_i} = \frac{\|(\mathbf{p}_{center} - \mathbf{v}_i) \times \mathbf{e}_i\|}{\|\mathbf{e}_i\|} \tag{4.35}$$

The edge equation is shown in (4.36). All the points lying into the edge have $0 \leq \alpha \leq 1$. In order to compute the parameter α for $\mathbf{p}_{\mathbf{e}_i}$, the procedure is shown in (4.37):

$$\mathbf{P} = \mathbf{v}_k + \alpha_P \mathbf{e}_i \tag{4.36}$$

$$\mathbf{p}_{\mathbf{e}_i} = \mathbf{v}_k + \alpha \mathbf{e}_i \tag{4.37}$$

$$\mathbf{p}_{\mathbf{e}_i} - \mathbf{v}_k = \alpha \mathbf{e}_i \tag{4.38}$$

$$(\mathbf{p}_{\mathbf{e}_i} - \mathbf{v}_k)\mathbf{e}_i = \alpha \mathbf{e}_i \mathbf{e}_i \tag{4.39}$$

$$\alpha = \frac{(\mathbf{p}_{\mathbf{e}_i} - \mathbf{v}_k)\mathbf{e}_i}{\|\mathbf{e}_i\|} \tag{4.40}$$

In order to fully define the contact, the direction for the contact must be established. It is defined as the vector pointing from $\mathbf{p}_{\mathbf{e}_i}$ to the center of the sphere, as shown in Figure 4.15.

Depending on the degree of exactitude requested for the contact method, there exist different procedures for dealing with multiple, simultaneous contacts. In some cases, the simplification of only taking one of the polygons into account is sensible, specially if their orientation is similar. In other occasions, the polygon for which the indentation is the largest is chosen. This is commonly done for forces whose full precision magnitude is not needed. That could be the case for automobile simulators, where usually the meshes defining the roads are coarse, and the reaction forces over the wheels are not changing too wildly as the supporting soil is smooth. In simulators where introducing impact forces resulting from crashes with the scenery is desired, it is also usually enough to consider only one contact point per sphere.

For the computation of collision forces against fine-grained meshes, all the polygons in contact must be considered for the force computations. If a force is introduced per contact point, a weighting algorithm must be devised in order distribute the forces, thus avoiding to accumulate force values within common directions to each reaction.

4.6.3. Mesh-Cylinder

This is a special case of collision detection with a shape expressed as an analytic function. As shown before, the main advantage of this kind of primitives is that they provide the exact definition of the surface. A polygonal discretization of a surface can be also computed for any arbitrary resolution level, but it requires a discretization pre-processing step and much higher memory requirements for its storage. If the surface of the object can be approximated as the composition of several primitives, this method can be very effective. For example, the cylinder primitive is well suited for describing many mechanical elements, so studying this particular case can be rewarding in terms of speed and accuracy.

Nonetheless, as the complexity of the analytic expression increases, the complexity of the collision detection algorithm grows as well. The capped cylinder is a defined as a quadratic form, and its intersection with a generic polygon cannot easily be found analytically. Numerical techniques must be used in order to find the region of common points to both shapes. Those points will fulfill the premise that the distance between them and the other surface is null. Therefore, the existence of the contact can be proved if at least a point is found to have zero distance. The problem can be formulated in terms of an optimization with inequality constraints. The equations are written in the cylinder space because the expressions of the cylinder are more complex than the polygon's.

The cylinder's coordinate frame and its definition parameters are depicted in Figure 4.16. The origin of the coordinate frame is placed at the center of the cylinder, and its z axis coincides with its directrix. The length of the cylinder is $2h$, and its radius, R.

Without loss of generality, the constraints presented are for the case that the polygon is a triangle. The triangle is defined by its three vertices, $\mathbf{p_0}$, $\mathbf{p_1}$ and $\mathbf{p_2}$,

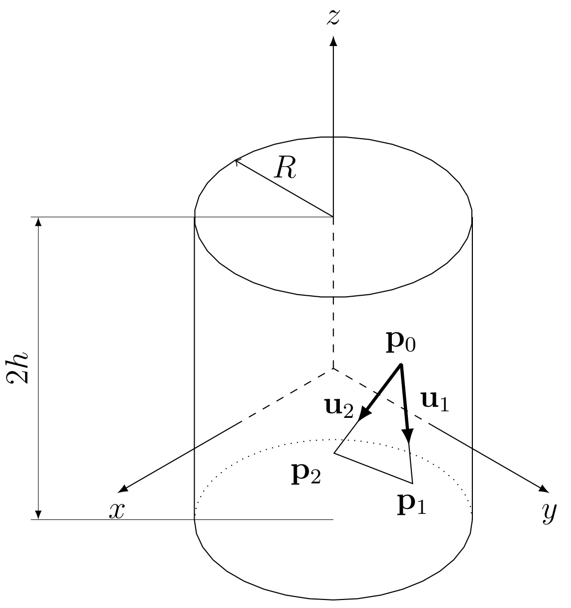


Figure 4.16: The cylinder-triangle contact problem.

and the unit vectors for the two edges that start from $\mathbf{p_0}$: $\mathbf{u}$ and $\mathbf{v}$.

The equations of triangle p^i:

$$\begin{gathered}
\mathbf{r}_t = \mathbf{p}_0 + \mu_1 \mathbf{u}_1 + \mu_2 \mathbf{u}_2 \left\{ \begin{array}{l} \dfrac{\mu_1}{l_1} + \dfrac{\mu_2}{l_2} \leq 1 \\ \mu_1 \geq 0 \\ \mu_2 \geq 0 \end{array} \right\} \\
\mathbf{u}_1 = \frac{\mathbf{p}_1 - \mathbf{p}_0}{l_1};\ \mathbf{u}_2 = \frac{\mathbf{p}_2 - \mathbf{p}_0}{l_2} \\
l_1 = \|\mathbf{p}_1 - \mathbf{p}_0\|;\ l_2 = \|\mathbf{p}_2 - \mathbf{p}_0\|
\end{gathered} \tag{4.41}$$

The optimization problem becomes:

$$\text{min. } r_p^2 = p_x^2 + p_y^2 \tag{4.42}$$

st.

$$\phi_1 = 1 - \frac{\mu_1}{l_1} - \frac{\mu_2}{l_2} \geq 0 \tag{4.43}$$

$$\phi_2 = h - p_z \geq 0 \tag{4.44}$$

$$\phi_3 = h + p_z \geq 0 \tag{4.45}$$

$$\phi_4 = \mu_1 \geq 0 \tag{4.46}$$

$$\phi_5 = \mu_2 \geq 0 \tag{4.47}$$

From (4.41)

$$p_x = p_{0x} + \mu_1 u_x + \mu_2 v_x \tag{4.48}$$

$$p_y = p_{0y} + \mu_1 u_y + \mu_2 v_y \tag{4.49}$$

$$p_z = p_{0z} + \mu_1 u_z + \mu_2 v_z \tag{4.50}$$

Equation (4.42) is the distance between a point of the triangle and the cylinder directrix; equations (4.44) and (4.45) are the boundaries of the cylinder while (4.43) (4.46) and (4.47) are the boundaries of the triangle. Note that the unknowns of the problem are $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$.

The Lagrangian of this problem is like follows.

$$L(\boldsymbol{\mu}, \boldsymbol{\lambda}) = r_p^2 - \lambda_1\phi_1 - \lambda_2\phi_2 - \lambda_3\phi_3 - \lambda_4\phi_4 - \lambda_5\phi_5 = r_p^2 - \boldsymbol{\phi}^{\mathrm{T}}\boldsymbol{\lambda} \tag{4.51}$$

The Karush-Kuhn-Tucker conditions provide the necessary conditions for the optimum.

$$\boldsymbol{\nabla}_{\boldsymbol{\mu}} L(\boldsymbol{\mu}, \boldsymbol{\lambda}) = 0 \tag{4.52}$$

$$\boldsymbol{\nabla}_{\boldsymbol{\lambda}} L(\boldsymbol{\mu}, \boldsymbol{\lambda}) \geq 0 \tag{4.53}$$

$$\lambda_i \geq 0; \quad i = 1, ..., 5 \tag{4.54}$$

$$\lambda_i\phi_i = 0; \quad i = 1, ..., 5 \tag{4.55}$$

The previous conditions are not easy to manage directly but they allow to transform the problem to several problems with equality constraints, instead of the inequality ones. Equations 4.55 say that the slack constraints have null multipliers and therefore they may be removed, solving only for the active constraints. After solving is necessary to check the conditions 4.53 and 4.54 to discover if it is necessary to activate or deactivate new constraints.

Since (4.42) is a quadratic function of the variables and the constraints are linear, the problem can be stated as a quadratic programming problem.

$$\left.\begin{array}{ll} \text{min.} & \frac{1}{2}\boldsymbol{\mu}^{\mathrm{T}}\mathbf{G}\boldsymbol{\mu} + \boldsymbol{\mu}^{\mathrm{T}}\mathbf{d} \\ st. & \mathbf{A}^*\boldsymbol{\mu} = \mathbf{b}^* \end{array}\right\} \Rightarrow$$

$$\begin{bmatrix} \mathbf{G} & -\mathbf{A}^{*\mathrm{T}} \\ \mathbf{A}^* & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\mu} \\ \boldsymbol{\lambda}^* \end{bmatrix} = \begin{bmatrix} -\mathbf{d} \\ \mathbf{b}^* \end{bmatrix} \tag{4.56}$$

Where.

$$\mathbf{G} = \begin{bmatrix} 2\left(u_x^2 + u_y^2\right) & 2\left(u_x v_x + u_y v_y\right) \\ 2\left(u_x v_x + u_y v_y\right) & 2\left(v_x^2 + v_y^2\right) \end{bmatrix} \tag{4.57}$$

$$\mathbf{d} = \begin{bmatrix} 2p_{0x}u_x + 2p_{0y}u_y \\ 2p_{0x}v_x + 2p_{0y}v_y \end{bmatrix} \tag{4.58}$$

$$\mathbf{A} = \begin{bmatrix} -\frac{1}{l_1} & -\frac{1}{l_2} \\ -u_z & -v_z \\ u_z & v_z \\ 1 & 0 \\ 0 & 1 \end{bmatrix} ; \ \mathbf{b} = \begin{bmatrix} -1 \\ -h + p_{0z} \\ -h - p_{0z} \\ 0 \\ 0 \end{bmatrix} \tag{4.59}$$

Note that $\mathbf{A}$ and $\mathbf{b}$ are the complete sets of constraints, while $\mathbf{A}^*$ and $\mathbf{b}^*$ are only the active sets. It was said before that after solving each problem with the current active set, it is necessary to check the validity of the set, adding or removing constraints if necessary. For this purpose active-set methods can be employed (see, for example [84]).

The solution to the problem is the vector $\boldsymbol{\mu}$, which determines the point inside the triangle at the minimum distance to the axis of the cylinder. The actual distance from its surface is calculated like follows, by subtracting the radius R of the cylinder.

$$\delta = r_p - R \tag{4.60}$$

4.6.4. Mesh-Torus

This is a similar case as the cylinder's. A torus shape can be used to represent bent metallic parts made from wires or threads. First, the nearest point from the polygon to its surface is found. Then, a selective process can be started if the contact over the whole torus is not desired: the point can be tested for inclusion into a pre-selected torus arc.

For the generic case of the semi-toroid, an analogous approach can be established: let's suppose the overlap test for the same triangle p^i, of §4.6.3, and a semi-toroid shown in Figure 4.17. The minimum distance between the triangle and the semi-toroid is given by the following constrained optimization problem with inequality constraints.

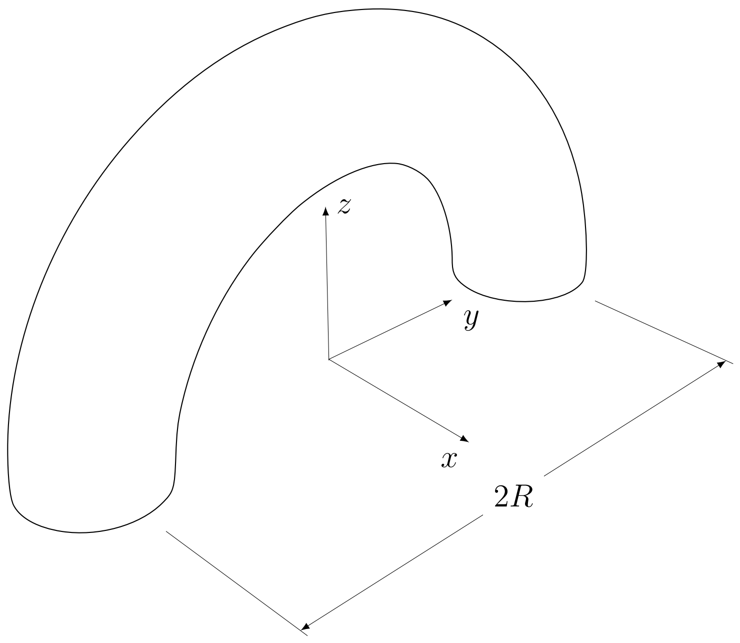


Figure 4.17: Semi-toroid definition. Its minor radius is r.

$$\text{min. } r_p^2 = p_x^2 + \left(\sqrt{p_y^2 + p_z^2} - R\right)^2 \tag{4.61}$$

st.

$$\phi_1 = 1 - \frac{\mu_1}{l_1} - \frac{\mu_2}{l_2} \geq 0 \tag{4.62}$$

$$\phi_2 = p_z \geq 0 \tag{4.63}$$

$$\phi_4 = \mu_1 \geq 0 \tag{4.64}$$

$$\phi_5 = \mu_2 \geq 0 \tag{4.65}$$

Equation (4.61) is the distance between a point of the triangle and the toroid directrix, equation (4.44) is the boundary of the semi-toroid while (4.43) (4.46) and (4.47) are the boundaries of the triangle. The unknowns of the problem are again $\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$.

The problem here is very similar to the cylinder overlap, with the difference that the objective function is not quadratic anymore. The technique is the same, to transform the inequality constraints to equality constraints and to use the active set method (see [84]).

$$\boldsymbol{\nabla}_{\boldsymbol{\mu}} L\left(\boldsymbol{\mu}, \boldsymbol{\lambda}\right) = \boldsymbol{\nabla}_{\boldsymbol{\mu}}\left(r_p^2\right) - \boldsymbol{\phi}_{\boldsymbol{\mu}}^{\mathrm{T}}\boldsymbol{\lambda} = \mathbf{0} \tag{4.66}$$

$$\boldsymbol{\phi}^* = \mathbf{0} \tag{4.67}$$

The problem can be solved using the Augmented Lagrangian technique.

$$\boldsymbol{\nabla}_{\boldsymbol{\mu}}\left(r_p^2\right) - \boldsymbol{\phi}_{\boldsymbol{\mu}}^{\mathrm{T}}\left(\,\boldsymbol{\lambda} - \alpha\boldsymbol{\phi}\right) = \mathbf{0} \tag{4.68}$$

Since $\boldsymbol{\nabla}_{\boldsymbol{\mu}}\left(r_p^2\right)$ is non-linear, equation (4.68) can be solved by means of the Newton-Raphson iteration.

$$\left[\boldsymbol{\nabla}_{\boldsymbol{\mu}\boldsymbol{\mu}}\left(r_p^2\right) + \boldsymbol{\phi}_{\boldsymbol{\mu}}^{\mathrm{T}}\alpha\boldsymbol{\phi}_{\boldsymbol{\mu}}\right]_j \Delta\boldsymbol{\mu}_{j+1} = -\left[\boldsymbol{\nabla}_{\boldsymbol{\mu}}\left(r_p^2\right) - \boldsymbol{\phi}_{\boldsymbol{\mu}}^{\mathrm{T}}\left(\,\boldsymbol{\lambda}_i - \alpha\boldsymbol{\phi}\right)\right]_j \tag{4.69}$$

The system (4.69) has to be solved iteratively keeping constant the values of $\boldsymbol{\lambda}_i$ until convergence. Once the convergence is attained, the outer iteration for the multipliers is performed.

$$\boldsymbol{\lambda}_{i+1} = \boldsymbol{\lambda}_i - \alpha\boldsymbol{\phi} \tag{4.70}$$

Taking derivatives to the equations of the objective function and constraints.

$$\boldsymbol{\nabla}_{\boldsymbol{\mu}}\left(r_p^2\right) = \left[\frac{\partial r_p^2}{\partial \mathbf{p}}\frac{\partial \mathbf{p}}{\partial \boldsymbol{\mu}}\right]^{\mathrm{T}} = \left[\begin{array}{ccc} u_x & u_y & u_z \\ v_x & v_y & v_z \end{array}\right]\left[\begin{array}{c} 2p_x \\ 2k_1p_y \\ 2k_1p_z \end{array}\right] = \frac{\partial \mathbf{p}}{\partial \boldsymbol{\mu}}^{\mathrm{T}}\frac{\partial r_p^2}{\partial \mathbf{p}}^{\mathrm{T}} \tag{4.71}$$

$$k_1 = 1 - \frac{R}{\sqrt{p_y^2 + p_z^2}} \tag{4.72}$$

$$\boldsymbol{\nabla}_{\boldsymbol{\mu}\boldsymbol{\mu}}\left(r_p^2\right) = \left[\frac{\partial^2 r_p^2}{\partial \mathbf{p}\partial\boldsymbol{\mu}}\frac{\partial \mathbf{p}}{\partial \boldsymbol{\mu}} + \frac{\partial r_p^2}{\partial \mathbf{p}}\frac{\cancel{\partial^2 \mathbf{p}}}{\cancel{\partial \boldsymbol{\mu}^2}}\right]^{\mathrm{T}} = \left[\begin{array}{cc} \frac{\partial \mathbf{p}}{\partial \boldsymbol{\mu}}^{\mathrm{T}}\frac{\partial^2 r_p^2}{\partial \mathbf{p}\partial\mu_1}^{\mathrm{T}} & \frac{\partial \mathbf{p}}{\partial \boldsymbol{\mu}}^{\mathrm{T}}\frac{\partial^2 r_p^2}{\partial \mathbf{p}\partial\mu_2}^{\mathrm{T}} \end{array}\right] \tag{4.73}$$

$$\frac{\partial^2 r_p^2}{\partial \mathbf{p} \partial \mu_1}^{\mathrm{T}} = 2 \begin{bmatrix} u_x \\ R\left(\frac{p_y u_y + p_z u_z}{k_2^3}\right) p_y + k_1 u_y \\ R\left(\frac{p_y u_y + p_z u_z}{k_2^3}\right) p_z + k_1 u_z \end{bmatrix} \tag{4.74}$$

$$\frac{\partial^2 r_p^2}{\partial \mathbf{p} \partial \mu_2}^{\mathrm{T}} = 2 \begin{bmatrix} v_x \\ R\left(\frac{p_y v_y + p_z v_z}{k_2^3}\right) p_y + k_1 v_y \\ R\left(\frac{p_y v_y + p_z v_z}{k_2^3}\right) p_z + k_1 v_z \end{bmatrix} \tag{4.75}$$

$$k_2 = \sqrt{p_y^2 + p_z^2} \tag{4.76}$$

$$\boldsymbol{\phi}_{\boldsymbol{\mu}} = \begin{bmatrix} \frac{-1}{l_1} & \frac{-1}{l_2} \\ u_z & v_z \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.77}$$

Note that $\boldsymbol{\phi}$ and $\boldsymbol{\phi}_{\boldsymbol{\mu}}$ are the complete sets of constraints and jacobian matrix, while $\boldsymbol{\phi}^*$ and $\boldsymbol{\phi}_{\boldsymbol{\mu}}^*$ are only the active sets.

The solution to the problem is the vector $\boldsymbol{\mu}$, which determines the point inside the triangle at the minimum distance to the semi-toroid. The distance δ is calculated like follows, being r the minor radius of the semi-toroid (the radius of its section).

$$\delta = r_p - r \tag{4.78}$$

4.6.5. Mesh-Chain link

This is an example of what can be accomplished by combining some of the methods described earlier. When possible, a three-dimensional object can be

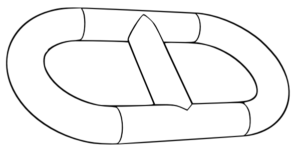


Figure 4.18: A chain link modeled as the union of two semi-toroids and three cylinders.

represented by the set of the union of several primitives that can be described with analytical expressions.

For example, a chain link can be modeled as the union of two semi-toroids and three cylinders, as shown in Figure 4.18. The advantage of using this method over a plain polygonal mesh lies in the absence of the need of having to set any resolution parameter in order to have the best possible accuracy.

At the first detection stage, the bounding boxes for every one of the primitives (cylinders and semi-toroids) is checked against the other potentially-colliding objects. If matching primitives are found, finer grained tests, specific to their nature, can be performed individually. Those specific primitive tests were described in §4.6.3 and §4.6.4. For each primitive in contact, the point, the normal vector of the surface at that location and the indentation is computed. Those are required parameters for contact force models discussed in Chapter 3.

4.7. Mesh-Mesh detection

The contact detection between meshes is the most versatile of the ones described in this document. The polygonal mesh describing any 3D object can be placed into the simulation, with at most minor modifications from the CAD program used to design it, obtaining a realistic behavior without further manual preprocessing steps. Polygonal meshes can approximate almost any three dimensional geometry. The attained precision degree depends on several factors: the density of the mesh and the degeneracy of its polygons.

A denser mesh can better approximate the shape of a complex curved surface. On the other hand, increasing the polygon count for an object raise as well the memory requirements for storing it into the computer, the number of primitive collision tests needed to be performed, and usually decreases the size of the polygons. The size of the polygons can constitute a problem if they are several orders of magnitude smaller than the rest of the primitives. Since the precision of the machine is limited, noticeable numerical errors can show up when performing calculations between numbers very different in magnitude.

Meshes with degenerate or very irregular polygons can also be the source of numerical errors and inaccuracies: parameters extracted from computations with the vertices of the polygons and primitive tests tend to be very sensible to numerical issues. For example, computing the normal vector of any triangle in the mesh is usually done with the cross product of two of the vectors representing its edges. If one of them is much more smaller than the other two, the cross product will result in a normal vector suffering from a high numerical error, thus having an incorrect orientation.

Let's propose a study on a degenerate triangle represented by three vectors as its edges in counter-clockwise order, $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$. It is clear that $\mathbf{u}-\mathbf{v}+\mathbf{w} = \mathbf{0}$. If $\|\mathbf{w}\| \ll \|\mathbf{u}\|$ and $\|\mathbf{w}\| \ll \|\mathbf{v}\|$, and the normal vector is computed as

$$\mathbf{n} = \frac{\mathbf{u} \times \mathbf{v}}{\|\mathbf{u} \times \mathbf{v}\|} \tag{4.79}$$

The expression for the cross product (4.80) reveals that, for each component, the *catastrophic cancellation* phenomenon [45] shows up if $\mathbf{u}$ and $\mathbf{v}$ have almost equal components. When subtracting similar floating point quantities, a significant loss of accuracy is produced, because the result is canceled except for the last digits. After the normalization of the vector, it can be possible that it is no longer perpendicular to the original facet.

$$\mathbf{u} \times \mathbf{v} = \underbrace{(u_j v_k - u_k v_j)}_{\approx 0}\,\mathbf{i} - \underbrace{(u_i v_k - u_k v_i)}_{\approx 0}\,\mathbf{j} + \underbrace{(u_i v_j - u_j v_i)}_{\approx 0}\,\mathbf{k} \tag{4.80}$$

The solution for those two problems can be alleviated by the use of special mesh generators. Finite Element Method mesh generators deal with this specific kind of problems. Their output is a mesh surface with almost regular elements — usually triangles —. Several of those mesh generators can reduce the size of the facets at the most complex areas of the object in order to achieve a high degree of accuracy, and increase it again for the simplest zones of the surface, therefore reducing the total polygon count. FEM meshes can be usually obtained from the Boundary Representation (BRep) [101] or the Constructive Solid Geometry (CSG) [79] representations used by CAD software, therefore attaining high levels of fidelity with respect to the original object. This is the case for the NETGEN software [95].

The disadvantage of the FEM mesh generators is the high number of triangles that are created during the discretization of the surface. It could result in memory storage and computational penalizations, since very dense meshes require a higher number of polygon tests. Consider the meshes presented in Figure 4.19: a FEM mesh and a visualization triangularization mesh are presented. The quality of the FEM mesh triangles is superior, but its density is much higher, even

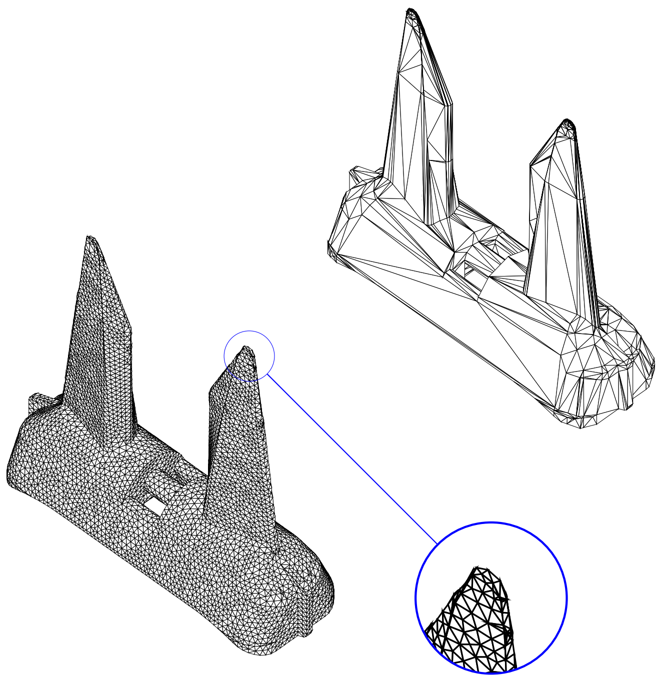


Figure 4.19: The upper right object represents the display triangularization which is usually obtained from CAD sfotware, while the bottom left model shows the output of a FEM meshing software.

serious efforts are made to diminish the number of additional vertices that the discretization implies [98].

The contact tests and modification algorithms are very generic, although elaborate. For example, a contact test will look for intersecting triangle pairs: first, the two AABB trees of both objects are tested for potentially colliding pairs of primitives; then, each pair is tested for interference. If any additional parameter is needed for applying a contact method, further processing must be done with the pairs, in order to deduce the contact point and the normal for the to-be-applied reaction forces.

Usually, the meshes are required to be manifold. In this context, a surface being manifold means that the mesh has to define a closed surface which defines volume. Thus, its surface cannot have any holes; any edge has to be shared exactly by two polygons. This is equivalent to require that the geometry cannot have *zero thickness* surfaces. This requirement implies the intersection lines of two objects always form closed contours. In addition, the intersection will also always define a closed volume. Notwithstanding, open surfaces can be used as well for collision purposes if care is taken in order to avoid collisions near the boundaries of the open surface. As an example, an “infinite” plane could be used as a ground mesh if the rest objects are small and far from its edges.

4.7.1. Topological information

Aside from the creation of the AABB trees for storing the structuring of the primitives of each object, there are more features than can be computed at an initial stage, since they are going to remain unmodified. These features can be used to simplify the run-time collision algorithms and to boost their performance as well.

It is useful to have a per object database containing the topological characteristics of the three dimensional meshes. As described before, it is usually needed to determine what should be the contact point of the intersection of two meshes. The closed contours described by the intersection of both need to be computed. To do so, it is needed to know which polygons are adjacent to a given one in order to find the line segments belonging to the contour. The information about the adjacent polygons for each primitive can be stored before, thus saving computational resources at run-time. Note that the topological information of the mesh remains unmodified even if shape of the mesh is modified in any way: scaled, distorted, rotated... The only event that can invalidate the neighboring database is the addition or removal of facets.

Some additional operations with two intersecting mesh objects can benefit from additional topological information. In addition to the adjacent facets for each of the edges of a polygon, it is also valuable to have a directory where information about those edges and their vertices is stored. Usually a mesh is defined by the list of its vertices, and the list of faces, defined by the indices of the vertices. This additional information is needed, for example, to be able to do an ordered walk over the vertices of a facet, or to iterate over all the edges that share a common vertex. Those iterations can be useful for computing at run-time the averaged normal for an edge or a vertex. Algorithms that subdivide the mesh at run-time can also benefit from the pre-computed information, e.g. volume intersection computation.

Research in Computer Graphics brought several adjacency information models. Their target is to provide adjacency information for a facet in constant time, therefore avoiding to do explicit searches at simulation time. The most popular ones are the *winged edge* [3], the *half edge* [79] and the *winged triangle* [105]. Those structures relate a polygon's feature with the ones from the neighboring facets. Their drawback consists on the extra memory consumption that they bring; typically this information can surpass the memory requirements of the mesh definition. There are additional side-effects to the extra memory allocation, as explained later. Therefore, an equilibrium must be found between the resource and performance implications of the higher memory requirements, and

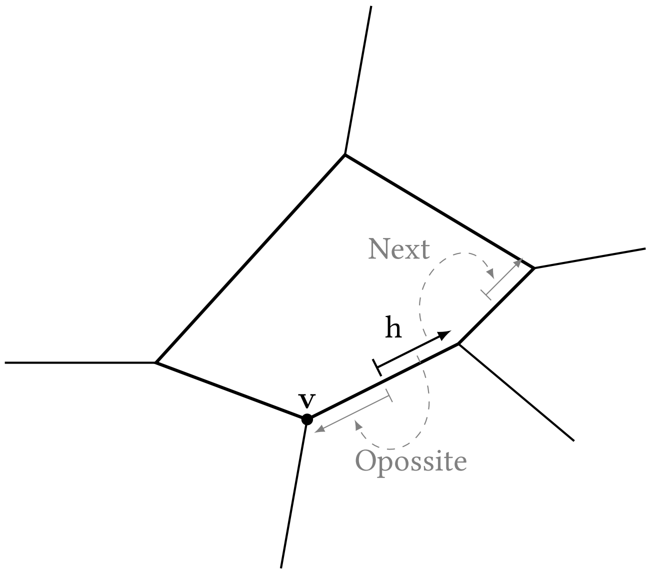


Figure 4.20: The *half edge* structure.

the saved computation time at simulation time.

The winged edge structures store local information about an edge: the vertices it joins, the faces it belongs to, and the four edges connected to it at both of its ends. Following the structure, the surface can be traversed by locating adjacent edges referenced by the current node.

The half edge structure (Figure 4.20) aims to have a smaller size, but preserving its usefulness: a node represents an edge of the mesh. It is called *half edge* because the edge is directed following the face vertex order (usually counterclockwise). If the mesh is manifold, a complementary half edge with the opposite directed is guaranteed to exist in the neighbor face. Each node holds references to its origin vertex, the following edge in the polygon it belongs to, and the node for the opposite or complementary half edge.

When stored in an array, each half edge node can hold as little information as just the polygon name to whom it belongs and the reference to its complementary node. If the number of sides of all the faces is the same, even the polygon reference can be discarded from the structure, since it can be deducted from the position of the node in the whole array. The performance implications can vary wildly depending on the size and nature of the data sets and the hardware where it is processed.

Finally, the winged triangle structure provides all the possible information about each face: its vertices, edges and adjacent faces. Those are big structures that, consequently, have big chances of be penalized because of their memory footprint when dealing with large amounts of data.

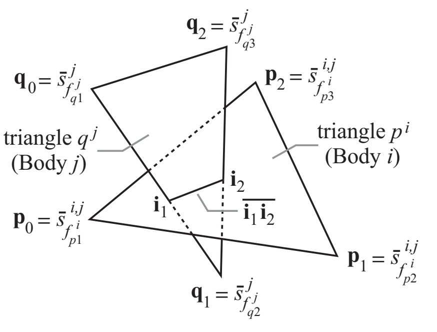
Hardware performance implications

Before, there were mentions to performance penalties showing up as larger data sets are used. From a hardware point of view, retrieving values from memory can have performance implications [31]. When the processor of a computing system has to read or write data from or to memory, a mechanism has to be activated in order to perform the transfer. Usually, data transfer over the channels that communicate the CPU and the RAM — called buses — is several orders of magnitude slower than any command that the CPU could issue. To alleviate this problem, data transfers are *buffered*. The use of a data buffer implies that the data is read or written in chunks: if the execution of a command in the CPU demands a certain value stored in the memory, then, not only that value, but the rest of the data — until the buffer is filled — is read. Therefore, in the very probable event that, at later computations, the CPU needs neighboring values to the last used, there also exist high chances of it already being stored into the buffer, thereby avoiding a data transfer. Each of the buffers is called a *cache line*, and all of them form the *cache* of the processor. Additionally, processors can have several levels of cache memories, ranging from the largest and slowest (high level numbers), to the smallest and fastest (small level numbers).

The key for exploiting the cache mechanism performance is to try as much as possible to work with data stored within a spatial locality; when working with arrays, for example, it is advisable to process them in order, as adjacent data values are likely to already be stored in the cache. Another rule is to try to design smaller data structures: the smaller they are, the more will fit into the cache, therefore increasing the chances for a given value to exist in the cache. When a value is not found in the cache, its whole contents are discarded, and a transfer from memory operation is begun. The failing event is called a *cache miss*.

Therefore, using simpler and smaller data structures for adjacency information storage can avoid hitting the memory transfer limitations of the hardware, maybe at the expense of having to perform extra computations. The global effect of the choosing of a structure over the available types has to be measured by means of performance tests: if there is a memory bandwidth bottleneck, there will be a high number of cache misses, whilst of the structure does not provide enough information, the program could be CPU-bound. Usually, as the size of the meshes' polygon count increases, the program is more likely to be memory-bandwidth limited, and smaller structures will perform better.

As reported before, as long as no polygon is added to or removed from the meshes, their adjacency structure remains immutable. This fact can be exploited in order to process several collision tests simultaneously, since there is no risk of falling into *race conditions* that usually appear in parallel programming. Multi-



$\mathbf{q}_2 = \bar{s}^{j}_{f^j_{q3}}$
$\mathbf{q}_0 = \bar{s}^{j}_{f^j_{q1}}$
$\mathbf{p}_2 = \bar{s}^{i,j}_{f^i_{p3}}$
triangle q^j
(Body j)
triangle p^i
(Body i)
$\mathbf{i}_2$
$\mathbf{i}_1$
$\overline{\mathbf{i}_1\mathbf{i}_2}$
$\mathbf{p}_0 = \bar{s}^{i,j}_{f^i_{p1}}$
$\mathbf{p}_1 = \bar{s}^{i,j}_{f^i_{p2}}$
$\mathbf{q}_1 = \bar{s}^{j}_{f^j_{q2}}$

Figure 4.21: Triangle-triangle intersection

processor systems can benefit from this fact, if each CPU core is dedicated to perform collision tests asynchronously.

4.7.2. Triangle-Triangle overlap algorithm

The tests for collisions between two mesh objects — made by testing their AABB trees against each other — end at each branch, when two intersecting leaves are found. Each leaf is a AABB box containing a triangle of the mesh, and therefore a test for checking triangle pair collisions must be carried.

Since triangle meshes are a widespread method for approximating and managing surface descriptions, there exist many algorithms for checking contacts between them. The *interval overlap method* [80], the *ERIT* package [62] or the *Orientation Predicates* method [55] are some examples. Those algorithms are very specific tests that try to minimize the number of necessary operations for checking the intersection of a pair of triangles. Unfortunately, for the purpose of computing reaction forces out of the position of the intersecting polygons, the points of intersection must be also calculated. Therefore the overall performance difference between the optimized methods and a series of edge-triangle intersections can be small in this case.

The algorithm described here is based on the direct solution of edge-triangle intersections, leading to a very robust behavior. The intersection of the triangles is typically a straight line segment, and this algorithm includes also the calculation of the extreme points of the segment, as this information will be required in following stages of the contact processing, see §4.7.3.

In Figure 4.21 the triangles p^i (from body *i*) and q^j (from body *j*), are intersecting. The vertices of a mesh are referred to the local frame of the mesh, e.g. f^j_{q1} is the first vertex of the triangle q of the body j, expressed in its local frame. The triangles are composed of the vertices $\mathbf{f}^i_p$ and $\mathbf{f}^j_q$ respectively. To check the intersection between them, it is enough to check each edge of triangle p^i against

q^j and vice versa. If the triangles overlap, two edge-triangle intersections exist. To illustrate the edge-triangle test, the intersection between edge $\overline{f^j_{q1} f^j_{q2}}$ and triangle p^i is calculated here.

Let's call $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ the vertices of triangle p^i and $\mathbf{q}_0$, $\mathbf{q}_1$, $\mathbf{q}_2$ the vertices of triangle q^j. First, both triangles must be expressed in the same coordinate frame. Using equation (4.81), it is possible to express all the vertices of the triangle p^i into the local frame of body *j*. Transformed coordinates are denoted by $\bar{s}$: for example, $\bar{\mathbf{s}}^{i,j}_{f^i_{p1}}$ is the first coordinate of the triangle p in body i, being transformed from the local frame of the body to the frame of the body j. s^i_0 and s^j_0 are the coordinates of the origin of the frames of the bodies i and j, expressed in the global frame of reference.

$$\begin{aligned}
\mathbf{p}_0 = \bar{\mathbf{s}}^{i,j}_{f^i_{p1}} &= \left(\mathbf{R}^j\right)^{\mathrm{T}} \left(\mathbf{s}^i_0 - \mathbf{s}^j_0\right) + \left(\mathbf{R}^j\right)^{\mathrm{T}} \mathbf{R}^i \bar{\mathbf{s}}^i_{f^i_{p1}} \\
\mathbf{p}_1 = \bar{\mathbf{s}}^{i,j}_{f^i_{p2}} &= \left(\mathbf{R}^j\right)^{\mathrm{T}} \left(\mathbf{s}^i_0 - \mathbf{s}^j_0\right) + \left(\mathbf{R}^j\right)^{\mathrm{T}} \mathbf{R}^i \bar{\mathbf{s}}^i_{f^i_{p2}} \\
\mathbf{p}_2 = \bar{\mathbf{s}}^{i,j}_{f^i_{p3}} &= \left(\mathbf{R}^j\right)^{\mathrm{T}} \left(\mathbf{s}^i_0 - \mathbf{s}^j_0\right) + \left(\mathbf{R}^j\right)^{\mathrm{T}} \mathbf{R}^i \bar{\mathbf{s}}^i_{f^i_{p3}}
\end{aligned} \tag{4.81}$$

$$\begin{aligned}
\mathbf{q}_0 &= \bar{\mathbf{s}}^j_{f^j_{q1}} \\
\mathbf{q}_1 &= \bar{\mathbf{s}}^j_{f^j_{q2}} \\
\mathbf{q}_2 &= \bar{\mathbf{s}}^j_{f^j_{q3}}
\end{aligned} \tag{4.82}$$

The equations of triangle p^i.

$$\begin{gathered}
\mathbf{r}_t = \mathbf{p}_0 + \mu_1 \mathbf{u}_1 + \mu_2 \mathbf{u}_2 \left\{ \begin{array}{l} \dfrac{\mu_1}{l_1} + \dfrac{\mu_2}{l_2} \leq 1 \\ \mu_1 \geq 0 \\ \mu_2 \geq 0 \end{array} \right\} \\
\mathbf{u}_1 = \frac{\mathbf{p}_1 - \mathbf{p}_0}{l_1};\ \mathbf{u}_2 = \frac{\mathbf{p}_2 - \mathbf{p}_0}{l_2} \\
l_1 = \|\mathbf{p}_1 - \mathbf{p}_0\|;\ l_2 = \|\mathbf{p}_2 - \mathbf{p}_0\|
\end{gathered} \tag{4.83}$$

The equations of edge $\overline{f^j_{q1} f^j_{q2}}$.

$$\begin{gathered}
\mathbf{r}_e = \mathbf{q}_0 + \eta \mathbf{v};\ 0 \leq \eta \leq d \\
\mathbf{v} = \frac{\mathbf{q}_1 - \mathbf{q}_0}{d};\ d = \|\mathbf{q}_1 - \mathbf{q}_0\|
\end{gathered} \tag{4.84}$$

Making $\mathbf{r}_t = \mathbf{r}_e$,

$$\begin{gathered}\mathbf{p}_0 + \mu_1\mathbf{u}_1 + \mu_2\mathbf{u}_2 = \mathbf{q}_0 + \eta\mathbf{v} \Rightarrow \\ [-\mathbf{v}\ \mathbf{u}_1\ \mathbf{u}_2]\begin{bmatrix}\eta \\ \mu_1 \\ \mu_2\end{bmatrix} = [\mathbf{q}_0 - \mathbf{p}_0] \Rightarrow \\ \mathbf{Ax} = \mathbf{b}\end{gathered} \tag{4.85}$$

There are 3 possible situations:

1. $\mathrm{rank}(\mathbf{A}) = 3 = \mathrm{rank}([\mathbf{A}|\mathbf{b}])$. The edge intersects the plane which contains the triangle. In case $\frac{\mu_1}{l_1} + \frac{\mu_2}{l_2} \leq 1$ with, $\mu_1, \mu_2 \geq 0$ and $0 \leq \eta \leq d$ the intersection lays into the triangle, otherwise the edge is discarded. In case of intersection, the intersection point can be easily calculated replacing η in (4.84).

2. $\mathrm{rank}(\mathbf{A}) = 2 = \mathrm{rank}([\mathbf{A}|\mathbf{b}])$. The edge is contained in the plane. The triangles might be coplanar or adjacent. The edge is discarded.

3. $\mathrm{rank}(\mathbf{A}) = 2 \neq \mathrm{rank}([\mathbf{A}|\mathbf{b}]) = 3$. The edge is parallel to the plane which contains the triangle. The edge is discarded.

First, the three edges of triangle q^j are successively checked against triangle p^i. Afterwards, the same operation is performed for the three edges of p^i against triangle q^j. In case that two intersections are obtained, the triangles overlap, and the intersection $\overline{\mathbf{i}_1\mathbf{i}_2}$ is given by the segment composed of the intersection points.

4.7.3. Collision pairs and contact regions

The output of the triangle-triangle detection phase is given as a list of colliding couples of polygons. Every couple consists of a pair of polygonal faces, each one belonging to either of the two objects being tested. A polygonal face from an object can show up several times in the list, since the relationship with the faces from the other object can be of the type “one to one”, “one to many” or “many to one”. As a convenience for later stages, the list of collision pairs is sorted in order to have all the collisions for a face grouped together. Obviously, an ordering can be found for either object. Every collision pair represents the intersection between two polygonal faces, and thus the segment defined by it.

Also, the ordering allows to follow the contours from a segment to the adjacent one. The algorithm uses the topological information about the neighbors, calculated in §4.7.1, to find out which collision pairs belong to the same contact region and to order the segments inside each region (4.87).

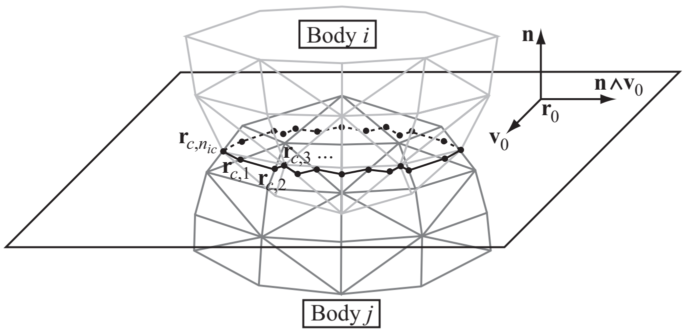


Figure 4.22: Contact plane calculation

$$\mathbf{pairs}^{ij} = \left\{ f_k^i, f_k^j \right\} ;\ 0 \le k \le n_p \text{ (list of pairs)} \tag{4.86}$$

$$\overline{\mathbf{isects}}^{ij,j} = \left\{ \overline{\mathbf{i}_{k,1}\mathbf{i}_{k,2}} \right\} ;\ 0 \le k \le n_p \text{ (list of segments)} \tag{4.87}$$

Once the segments are grouped by regions and the segments of each region are ordered, the algorithm merges the adjacent segments removing the coincident vertices in (4.87), by means of a simple numerical procedure. Finally, the algorithm returns a list with the existent n_c 3D contours given by their ordered vertices.

$$\begin{gathered} \overline{\mathbf{c}}_c^{ij,j} = \left[\begin{array}{cccc} \mathbf{r}_{c,1} & \mathbf{r}_{c,2} & \dots & \mathbf{r}_{c,n_{ic}} \end{array} \right] ; \\ 0 \le c \le n_c \text{ (list of contours)} \end{gathered} \tag{4.88}$$

In (4.88), n_{ic} is the number of vertices of the contour *c*, the super index ^{ij}, indicates collision between bodies *i* and *j*, and the over line along with the super-index ^{,j} indicates local coordinates of body *j*.

4.7.4. Contact plane

For each one of the contact regions identified in §4.7.3, the algorithm calculates the equations of the contact plane that better fits the 3D contour, (4.88), of the region (see Figure 4.22).

Replacing the vertices of the contour given by (4.88) in the equations of the contact plane.

$$\left[\begin{array}{cc} \mathbf{r}_{c,1}^{\mathrm{T}} & 1 \\ \mathbf{r}_{c,2}^{\mathrm{T}} & 1 \\ \dots & \dots \\ \mathbf{r}_{c,n_{ic}}^{\mathrm{T}} & 1 \end{array} \right] \left[\begin{array}{c} \overline{\mathbf{n}} \\ d \end{array} \right] = \left[\begin{array}{c} 0 \\ 0 \\ \dots \\ 0 \end{array} \right] \Rightarrow \mathbf{A}\mathbf{x} = \mathbf{0} \tag{4.89}$$

Where $\overline{\mathbf{n}}$ is a vector normal to the contact plane and $d = \ -\mathbf{r}_c^{\mathrm{T}}\overline{\mathbf{n}}$ being $\mathbf{r}_c$ a point that belongs to the contact plane.

In general the system of equations 4.89 has the only solution $\overline{\mathbf{n}} = \mathbf{0}; d = 0$, which obviously is not the desired solution. It is necessary to impose the condition $\|\overline{\mathbf{n}}\| = 1$ to obtain an incompatible system of equations that can be solved by least squares.

Writing the least squares system from 4.89.

$$\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)\mathbf{x} = \mathbf{0} \tag{4.90}$$

is either rank deficient or if it is full rank, the only possible solution is the trivial solution. Factoring the matrix $\left(\mathbf{A}^{\mathrm{T}}\mathbf{A}\right)$ and imposing the value (for example equal to 1) of the component of $\{\mathbf{x}\}$ corresponding to the minimum pivot of the factorization, the equations of the contact plane are obtained. Finally $\{\mathbf{x}\}$ has to be scaled to fulfill the condition $\|\overline{\mathbf{n}}\| = 1$, obtaining the final equations of the contact plane.

$$\overline{\mathbf{n}}^{\mathrm{T}}\mathbf{r} + d = 0 \tag{4.91}$$

Where $\overline{\mathbf{n}}$ is the unit normal vector to the plane and *d* is the distance from the plane to the origin measured along the normal vector.

All the calculations described in this section were performed with the contour of equation (4.88) expressed in the local reference frame of body *j*. The normal vector transformed to the global reference frame is obtained by means of the rotation matrix of body *j*.

$$\mathbf{n} = \mathbf{R}^{j}\overline{\mathbf{n}} \tag{4.92}$$

4.7.5. Contact centroid

For each one of the contact regions identified in §4.7.3, the algorithm described in this section, calculates the centroid of the projection of the contact region into the contact plane.

The centroid of a general 2D polygon of *N* vertices, contained in the $z = 0$ plane has the following expression.

$$\begin{gathered}\overline{\mathbf{r}}_c^{\triangle} = \frac{1}{6A}\sum_{i=1}^{N}\begin{bmatrix}\left(x_i + x_{i\oplus1}\right)\left(x_i y_{i\oplus1} - x_{i\oplus1}y_i\right)\\ \left(y_i + y_{i\oplus1}\right)\left(x_i y_{i\oplus1} - x_{i\oplus1}y_i\right)\\ 0\end{bmatrix}\\ A = \frac{1}{2}\sum_{i=1}^{N}\left(x_i y_{i\oplus1} - x_{i\oplus1}y_i\right)\end{gathered} \tag{4.93}$$

Nevertheless the contour, *c*, of equation (4.88) does not constitute a 2D polygon since its vertices do not belong, in general, to the same plane. To assimilate the contour to a 2D polygon, the vertices can be projected into the contact plane calculated in §4.7.4. Moreover, the resulting 2D polygon has to be contained in the $z = 0$ plane, what can be achieved by means of the transformation matrix $\mathbf{M}_t$, which transforms the $z = 0$ plane into the contact plane of equation (4.91).

The mentioned transformation matrix has the following expression.

$$\begin{aligned} \mathbf{M}_t &= \begin{bmatrix} \mathbf{M}_r & \bar{\mathbf{r}}_0 \\ 0 & 1 \end{bmatrix} \\ \mathbf{M}_r &= \begin{bmatrix} \bar{\mathbf{v}}_0 & \bar{\mathbf{n}} \times \bar{\mathbf{v}}_0 & \bar{\mathbf{n}} \end{bmatrix} \end{aligned} \tag{4.94}$$

Where $\bar{\mathbf{r}}_0$ and $\bar{\mathbf{v}}_0$ are a point and a vector contained in the contact plane, respectively, that can be chosen like follows.

$$\begin{aligned} \bar{\mathbf{r}}_0 &= \begin{bmatrix} -d/n_x \\ 0 \\ 0 \end{bmatrix} ; \bar{\mathbf{v}}_0 = \begin{bmatrix} -n_y \\ n_x \\ 0 \end{bmatrix} ; \quad n_x = \max(n_x, n_y, n_z) \\ \bar{\mathbf{r}}_0 &= \begin{bmatrix} 0 \\ -d/n_y \\ 0 \end{bmatrix} ; \bar{\mathbf{v}}_0 = \begin{bmatrix} n_y \\ -n_x \\ 0 \end{bmatrix} ; \quad n_y = \max(n_x, n_y, n_z) \\ \bar{\mathbf{r}}_0 &= \begin{bmatrix} 0 \\ 0 \\ -d/n_z \end{bmatrix} ; \bar{\mathbf{v}}_0 = \begin{bmatrix} n_z \\ 0 \\ -n_x \end{bmatrix} ; \quad n_z = \max(n_x, n_y, n_z) \end{aligned} \tag{4.95}$$

Being n_x, n_y, n_z the components of $\bar{\mathbf{n}}$. Expressing the contour in the local frame of the plane.

$$\bar{\mathbf{c}}_c^{ij,\triangle} = \mathbf{M}_r^{\mathrm{T}} \left(\bar{\mathbf{c}}_c^{ij,j} - \begin{bmatrix} \bar{\mathbf{r}}_0 & \bar{\mathbf{r}}_0 & \dots & \bar{\mathbf{r}}_0 \end{bmatrix} \right) \tag{4.96}$$

Replacing the *x* and *y* components of 4.96 in 4.93, the centroid $\bar{\mathbf{r}}_c^{\triangle}$, in local coordinates of the plane, is obtained.

Finally, the centroid expressed in global coordinates has the following expression.

$$\mathbf{r}_c^{ij} = \mathbf{s}_0^j + \mathbf{R}^j \left(\bar{\mathbf{r}}_0 + \mathbf{M}_r \bar{\mathbf{r}}_c^{\triangle} \right) \tag{4.97}$$

4.7.6. Maximum indentation

For each one of the contact regions identified in §4.7.3, the algorithm calculates the maximum indentation (or inter-penetration), δ. This algorithm travels

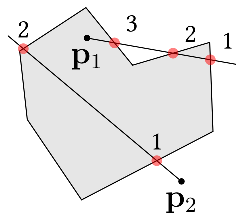


Figure 4.23: Ray-casting algorithm for determining if a point lies inside a polygon.

along the vertices $\mathbf{v}_i$ of the colliding triangles (4.86) and their neighbors inside the contact region, looking for the maximum indentation,(4.99). In order to distinguish the triangles of the first body that are inter-penetrating the second, it is necessary to check, for each neighbor not belonging to the colliding triangles list, two conditions: 1) the distance of each vertex of the triangle to the contact plane, calculated in §4.7.4, is negative and 2) the projection each one of the vertex of the triangle into the contact plane lies inside the projected contact region contour of §4.7.3 (see Figure 4.22). Between the lists of colliding and internal triangles of each body, the algorithm looks for the maximum indentation.

$$\delta_{\mathbf{v}_i} = \mathbf{n}^{\mathrm{T}}(\bar{\mathbf{s}}_i^{\triangle} - \bar{\mathbf{r}}_c^{\triangle}) \tag{4.98}$$

$$\delta = \max(\delta_{\mathbf{v}_i}),\ \forall \mathbf{v}_i \tag{4.99}$$

The checking of the condition 1) is straightforward while the condition 2) is checked by means of a ray casting algorithm: a ray with arbitrary direction, departing from a point is checked against the contour segments. If the number of intersections of is an even number, the point is found to be outside the polygon, and it is odd if the point is inside the polygon [102]. Figure 4.23 illustrates the algorithm for two points: $\mathbf{p}_1$ lies inside the polygon, and any ray cast from it will intersect the contour an odd number of times. Meanwhile, rays cast from $\mathbf{p}_2$ — being it in the exterior side of the contour — will have an even or null number of intersections.

The final indentation passed to the force model is the maximum value of the maximum indentation for each object.

4.8. Mesh-Convex Polyhedron detection

When the object to be checked for contact with the mesh is a convex polyhedron, some assumptions can be made in order to simplify the collision procedures.

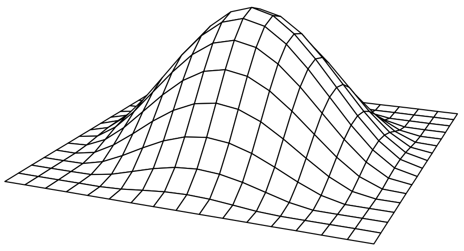


Figure 4.24: A regular mesh grid can be used to simulate deformable objects.

A polyhedron is defined as a set of flat faces enclosing a finite volume. A polyhedron is a convex volume if, for any pair of points pertaining to the volume, any intermediate points also belong to it.

$$\forall\, \mathbf{r}_i, \mathbf{r}_j \in \mathcal{P} \Rightarrow \mathbf{r}_\lambda = \mathbf{r}_i + \boldsymbol{\lambda}(\mathbf{r}_j - \mathbf{r}_i) \in \mathcal{P} \tag{4.100}$$

$$0 \leq \boldsymbol{\lambda} \leq 1 \tag{4.101}$$

where $\mathcal{P}$ denotes a convex polyhedron.

A polyhedron can be defined with the set of planes of its facets. Tests for finding points inside the polyhedron can be easily performed by checking if a point lies inside or outside of each of the planes. This is equivalent to the degenerate case shown in §4.4.2. The number of checks grows with the complexity of the polyhedron, so special algorithms have been developed in order to perform a minimal set of tests for collision purposes. For general purpose polyhedron-polyhedron collision testing, the GJK algorithm [43] is one of the most efficient, avoiding to test each point of one polyhedron against the other.

When a polyhedron is convex, a point can be quickly discarded as soon as it is found to be outside of any of its planes. Very simple and fast collision tests can be carried by surrounding geometric primitives by convex polyhedral shells, sometimes also referred as a *convex hulls*, provided that the polyhedron has a small number of facets.

4.8.1. Mesh deformation with convex hulls

Convex hulls or polyhedrons, and their quick discarding of points that lie outside of their volume, can be useful for some simulation tasks. In the following chapter, an excavator simulator will be presented. One of the features of the simulator is the ability of dig the terrain with the machine. Such a system needs: to be able to detect when the bucket of the machine is touching the terrain; how to deform it after the digging action.

A simple model can be crafted using convex hulls and regularly-spaced grid meshes. Using a convex hull for the bucket simplifies the task of determining which parts of the deformable terrain are in contact with it; the regular grid for the terrain eases the detection of the possible vertices touching the bucket, and their final placement, making make grooves or holes on its surface.

For regular mesh grids as the one shown in Figure 4.24, the computation of potentially affected facets is also simple, since a spatial ordering is assumed over the plane of the grid. Parts from the mesh intersecting with the polyhedral volume enclosing the tool geometry can be tested for collision purposes with the algorithm described previously. The mesh can be deformed by projecting the points that lie inside the polyhedron against the polyhedron facets.

An aligned bounding box for the bucket polyhedron hull has to be built in order to make the list of affected facets in the mesh patch. As seen in §4.5.1, the AABB can be computed from the current position of the vertices of the polyhedron. Should the need arise, for efficiency purposes, the AABB could be computed at an initial stage, and later just transformed by the current position and orientation of the bucket.

For each instant of time, the AABB for the bucket is calculated. The horizontal extension $[x_{min}, x_{max}]$ and $[y_{min}, y_{max}]$ of the AABB can be used to detect and group all the points from the patch mesh that lie inside that area. Those points are the ones that are potentially being modified by the bucket motion. If a vertex of the mesh is found to lie inside the convex hull, it is noted down in a list for later reference.

When the bucket moves, the points of the terrain mesh that are no longer inside must be detected and moved in order to reflect the marks of the bucket in the terrain surface. The list of points is searched for points that are outside the current boundaries of the bucket's polyhedron. If a vertex on that list is found to lie outside of the hull, it means that it should be displaced.

The displacements are only executed in the vertical direction, for preserving the grid shape. It is needed to compute how much the vertical coordinate of the vertex must be decremented so it is placed at the border of the hull (Figure 4.25).

For each of those points, a vertical ray is intersected with the polyhedron's facets, using the same equations (4.103). The resulting height of the intersection will define the new vertical position of the point. This is a reasonable approximation of the behavior of a clayey material, which preserves the shape of the tools used to dig into it.

Discharging maneuver

The discharging maneuver is aimed at simulating earthmoving techniques. Two procedures have been developed. Their aim is to determine when the mate-

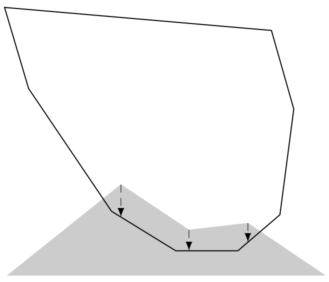


Figure 4.25: Vertices are projected vertically over the convex hull boundaries.

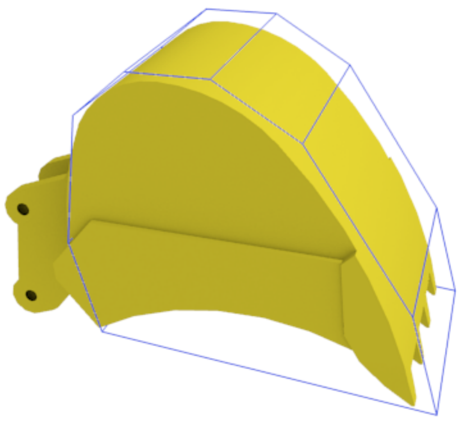


Figure 4.26: Original bucket object, and its convex hull (in blue, superimposed).

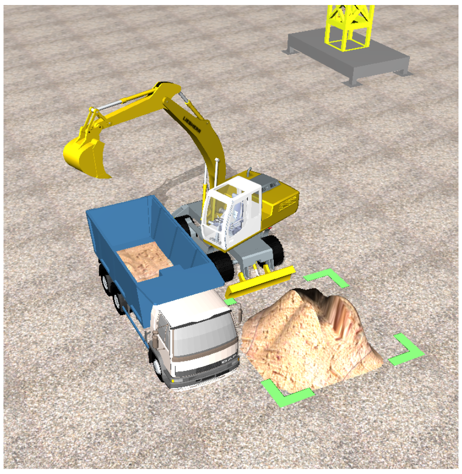


Figure 4.27: Digging operation.

rial should be removed gradually from the bucket, if its orientation lies between a range where it should be emptied, and to simulate the heaping of the material landing on top of existing surfaces or heaps.

The algorithm to decide whether the bucket should discharge material is very simple. The decision is taken looking at the value of the *discharging angle* β. It is considered that the bucket 4.26 is discharging its material when it lies in the range $-\frac{\pi}{2} < \beta < \frac{7}{9}\pi$, see Figure 4.28. The model computes the maximum material held into the bucket for the current orientation.

The discharge rate is deduced from the differences in the material volume between subsequent states. It can be estimated by a linear function that takes as its independent variable the spin speed of the bucket, $\dot{\beta}$, or in a practical way, $\Delta\beta/\Delta t$.

$$\nabla V = V_t - V_{t-1} \tag{4.102}$$

How to make a plausible representation of the heaping of the falling deposited material can be tackled in two steps: creating or adding new three-dimensional geometry to represent it, and how to distribute the material flow over those surfaces.

Regarding the creation or use of existing deforming meshes representing discharged material, a list of existing meshes can be used to know if it is already present a mesh into the discharge zone. Since usually there are too few of those meshes, they can be stored into a plain list, without having to resort to more complex solutions as required, for example, for collision detection purposes. Should a mesh patch is already placed into the discharge zone, the program can proceed to the discharge step. If not, a new mesh patch must be created and added into

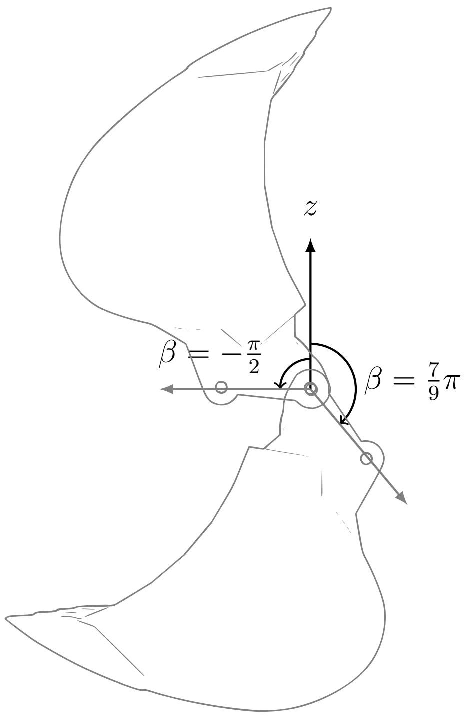


Figure 4.28: Discharge angle β.

the list.

Creation of new terrain discharge patch meshes

A new mesh can be generated as a regular, rectangular grid. The height of every point in the mesh is determined by the amount of the material dumped onto it. Usually, the place where the patch is placed is not flat. Thus, the patch must be modified in order to suit the shape of the target soil surface. This is done by adjusting every vertex point in the patch (p_x, p_y, p_z) to match the height of its vertical projection over the scenery surface, p_z.

This projection is a simple procedure that is carried for every point (p_x, p_y) in the patch mesh. A vertical ray passing through $(p_x, p_y, 0)$ is cast, and its intersection with the scenery is computed. The result is the triangle where the point $\mathbf{p}$ should rest. For The computation of the projection height of a point of the patch p_z over the triangle defined by vertices $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2$ is done as show in (4.103)

$$\mathbf{n} = (\mathbf{v}_1 - \mathbf{v}_0) \times (\mathbf{v}_2 - \mathbf{v}_0) \tag{4.103}$$

$$D = -\mathbf{n}\mathbf{v}_0 \tag{4.104}$$

$$p_z = -\frac{D + p_x n_x + p_y n_y}{n_z} \tag{4.105}$$

Terrain distribution

At each instant in the simulation where the bucket is discharging, a landing point is computed for the material. A parabolic trajectory is computed, starting from the bucket and dependent on the initial estimated velocity of the material flow. The intersection between the trajectory and the ground defines a target point (t_x, t_y).

In order to distribute the material being dropped over the mesh patch, a continuous gaussian probability distribution is used. A probability distribution alleviates the unrealistic effect that a predefined filling animation would show, by having a slightly different behavior each time it is used, yet describing a bell-shaped probability density function. Unfortunately, for most of the computing systems, if they provide any kind of random number generators, they are usually uniform density probability functions. Although uniform random generators are very convenient for many problems, gaussian probability distributions are better fitted when simulating natural phenomena.

There are several methods for converting random values from a uniform density function into a gaussian or normal one; they vary regarding their computing cost and reliability. It has been observed experimentally that, nevertheless, the computational cost of those algorithms is generally low compared with the rest of the components of the simulation program.

A procedure for obtaining a pair of gaussian random values g_x and g_y from a pair of uniform random values x_1 and x_2 is described in [71]. The algorithm is described as

Listing 4.1: A normal distribution random generator

```
do {
        x1 = 2.0 * ranf() - 1.0;
        x2 = 2.0 * ranf() - 1.0;
        w = x1 * x1 + x2 * x2;
 } while ( w >= 1.0 );

 w = sqrt( (-2.0 * ln( w ) ) / w );
 gx = x1 * w;
 gy = x2 * w;
```

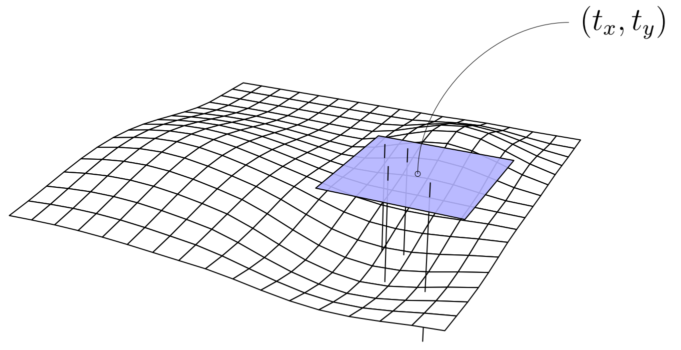


Figure 4.29: Ray casting method for distributing the flow over the mesh: a zone (in blue) is placed over the discharge zone. Then, random-positioned rays are cast from there to the mesh, determining the vertices to move.

In the initial loop, a pair of uniform random numbers in the range $-1 < x_i < 1$ are searched for, but the pair has to fulfill a condition so that their norm is lesser than one, $w = x_1^2 + x_2^2 < 1$. The module w is then transformed by the expression $w^* = \sqrt{-\frac{2\ln(w)}{w}}$. Finally, the gaussian conforming variables are computed from the original ones by multiplying them by the w^* factor:

$$g_x = x_1 w^* \tag{4.106}$$

$$g_y = x_2 w^* \tag{4.107}$$

The shape and location of the gaussian distribution is modified by adjusting its mean $\boldsymbol{\mu}$ and standard deviation $\boldsymbol{\sigma}$. Here, the mean $\boldsymbol{\mu} = (\mu_x, \mu_y) = (t_x, t_y)$ is the target point, so the random points are placed around it, and the $\boldsymbol{\sigma} = (\sigma_x, \sigma_y)$ is the parameter that controls how far from the target the points can be placed.

Terrain displacement

In order to increase the height of the mesh patch at the appropriate locations, a small rectangular area is defined, centered at (t_x, t_y). Several random points (g_{xi}, g_{yi}) are placed over the area, using the gaussian distribution. A ray is cast from each of those points in order to pierce the terrain mesh (Figure 4.29). After those intersections are computed, the affected polygons' vertices in the terrain mesh are raised by a quantity that depends on the terrain flow discharge and the density of the target mesh (4.108),

$$\Delta z = \frac{k \nabla V}{n_r \left(\frac{l_p}{n_c}\right)^2} \tag{4.108}$$

where n_r is the number of cast rays, l_p the size of the terrain patch, and n_c the number of cells along either of the axes of the patch. k is a proportionality constant that adjusts the discharge flow ∇V to the displacement.

Chapter 5

HiL Simulation

Certainly, one of the biggest advantages of the multibody techniques relies in the ability of introducing inputs of heterogeneous nature into the simulation. An useful case of study is the motion tracking of a real system in order be able of computing at a later stage the mechanical stresses supported by it, or the required efforts for performing that motion. Although those use cases are widely studied and, in fact, many specific methods exist, multibody techniques can be very valuable when it is not easy or feasible at all to obtain a simple characterization of the movement of the parts. Indeed, when experimental motion data such as the readings from sensors, or from optical systems can be obtained, it is very cumbersome to integrate them into analytic models without the use of software tailored for a single — or very reduced — specific use cases. Research fields as Biomechanics can benefit from multibody techniques, easing the characterization of the motion of human beings, since it is very difficult or intrusive to place precision sensors over or into a body [27].

However, there are numerous study cases where the inputs to the simulation are even more complex, and cannot be known before the start of the simulation because they are influenced by the nearly instantaneous state of the system. This is specially evident when considering mechanical systems whose input does not stem from a predefined law or model, but from a more complex controller: frequently, it can only be regarded as an opaque *black box*, whose operation is unknown. This is the case for inputs coming from a human operator or by a sufficiently complex hardware driver. The term *HiL* tries to include both cases, joining together the terms *Human in the Loop* and *Hardware in the Loop*.

Multibody systems are able to cope in a efficient way with this kind of heterogeneous inputs, in part because of their numerical nature. Time discretization allows for the simulator to send the current state to the input actors — being them of human nature or not — and later to receive an appropriate response in order to compute the next upgraded state. For example, human-driven machine simulators receive instantaneous inputs from the driver, which will depend on the current state of the simulator and the behavior of the driver. Hardware control systems can be tweaked or trained with a multibody counterpart of the real mechanism. Currently, there exists research about having multibody systems behave as a *virtual sensor*: given the readings of a few sensors in a real mechanical system, feed with them a multibody system that replicates the original mechanism [25]. Because the multibody system is synchronized with the original model, any measurement done in the multibody model would be equal to the real one, avoiding the requirement of placing a high count of sensors in the machine for control purposes.

In this document, several examples of real machine simulators are presented, showing the integration between a multibody system and input sensors that guide it at each time instant. Simulation sessions can be registered and recorded in order to check any aspect of the motion at will, therefore assisting in design or training tasks.

5.1. Input Peripherals

Input peripherals are those devices used to carry external information to the simulator program. Usually they are used to make the simulation aware of the commands issued by a user or an intelligent controller. In *Human in the Loop* simulations, the user is typically responsible for acting over the available controllers in order to command the system, given the stimulus that are received from the simulator. Other popular simulation kind is the *Hardware in the Loop* simulation, where a external system or piece of hardware is able to send either commands or sensor readings[1] to the simulator. This is generally useful to test the behavior of a control device over a real system too expensive or difficult to test. Of course, a *Human and Hardware in the Loop* solution can be also possible to implement.

In fact, considering the human or the hardware controllers as interchangeable parts of the whole simulator is a paradigm that is growing in popularity these days. At any time, any component of the system can be substituted by another one which performs the same tasks. Regarding human behavior, in the field of the Virtual Reality applications, the VRPN library [103] follows this principle: a client-server model where the simulation accepts commands from a server, which can be at the same time interacting with a bundle of heterogeneous input devices. The client-server model decouples the simulator from the controller or

[1]For example, readings from the sensors of a real automobile suspension *attached* to the virtual model of a vehicle.

input devices, allowing that this processing can take place in any other computing system, as long as a network connection exists between them.

When developing simulators running on standard PC computers, the easiest way of interfacing with a human is to use the commonly available input devices, such as the keyboard or the mouse. Not only the operating system fully supports them, but also those devices offer a reasonable refresh rate, enough for capturing a person's movements. On the other hand, these peripherals are not very similar to the controls that the simulators usually require — they should replicate levers, steering wheels, pedals, button panels, etc. — so analogical and high resolution inputs are usually required.

A improvement step in the direction of the replication of the real control devices used in the simulated system is the use of game controllers. This kind of devices are also well supported by operating systems, they are affordable and usually employed to simulate the same kind of machines that a realistic simulator tries to emulate. Among their downsides, its non-professional nature renders them as fragile and imprecise pieces of equipment, so they are not commonly seen in professional simulators. However, they constitute an excellent starting point for testing purposes in the development phases of the simulator.

Nowadays, there exist some industrial-quality controllers such as joystick levers and buttons that are close replicas or even just the same as the devices installed on real machinery and professional systems [89]. In Figure 5.1 it is shown an industrial lever controller which features Hall-effect sensors, which are very durable since they do not have contacting parts. In addition, those controllers usually implement standard ports, which can be read directly by the computers performing the simulation. The use of serial (RS-232) and USB ports aids in the integration process with the PC.

The serial port is a low-speed communication by today's standards, but nevertheless suitable for the data bandwidth required for this kind of task. Should the parameters for establishing a communication between the computer and the devices were known, the information can be read on a straightforward manner by means of simple calls to the operating system's API. In the common case that the computing system lacks a serial port available, there are a vast number of commercially available USB converters that allow to connect the device to a USB port. From the operating system perspective, the converter creates a *virtual serial port* that behaves as a real one.

For devices providing USB interfaces, the communication with the computing platform is even easier, since they usually identifies themselves as a HID (Human Interface Device) [109]. HID-USB is a special class in the USB communication standard which defines the identification and data transfer between the controller and the PC. Thus, the simulator can be designed independently of the exact model of controller used eventually, since the software can query it for the

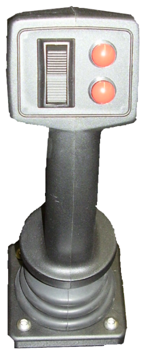


Figure 5.1: Hall effect, industrial quality joystick.

number of channels and buttons it provides. Each channel typically represents the measurement of the movement of a lever in one axis' direction, but it also can be used for other analogical controls, such as wheels. As a matter of fact, usually the HID drivers in modern operating systems present a resolution of 16-bit per channel, on par with many commercial sensors. However, this does not mean that the device is necessarily required to offer that resolution, only that the OS will present the readings in that range.

Finally, there are many controllers which do not implement any direct way of connection with a computing platform. This is in fact the case for the vast majority of industrial equipment employed in real machinery, which usually implements a wired, analogical signal output. This kind of equipment is meant to be connected directly to any other electronic devices, like relays or simpler board or micro-controller gates. In order to be able to read the data from these devices, other equipment is needed, typically a data acquisition device. There are a huge number of brands and models, being its features mainly determined by the number of channels they can sample, the maximum frequency at which the sampling is done, its resolution and its analogical or digital nature. Not many of those devices are really suitable for real-time purposes, since the communication between the data acquisition hardware and the computer suffers from great latencies. Cheapest hardware is aimed to sample and store several seconds or fractions of a second and then serve the data after that. This is commonly called a *trigger mode*, since the device waits for a signal —trigger—, after which it will capture a burst of samples and then transmit them back to the PC. If a single sample mode is provided, it is commonly implemented in terms of a burst mode, which will drop all the samples recorded except for the last one. In some cases, the single sample is implemented by channel, so trying to read several channels through this method would incur in as many times the latency penalty for each channel. Better data acquisition hardware presents a much lower sampling latency and resolution, but they typically cost thousands of euros. At the same time, this opens the possibility of entering any kind of signal any device can generate into the simulator, even integrating real devices in the simulator.

When using modest sampling hardware, the latency can be of the same order of magnitude or even greater than the time spent by the simulator on performing a time step. This means that the simulator would have to wait for the input, preventing it from reaching interactive or real-time frequency rates. Obviously, the sampling task will have to be isolated in a different flow of execution, either by using an alternate thread or another process. The downside lies in that an extra synchronization mechanism between both processing flows must be implemented, and that the input latency will be as high as it was before: the only goal was to avoid the sampling code to stall the whole simulation.

5.2. Graphical output devices

Graphical output is the primary way in which a simulator can reflect its state to its users. The sensitivity of the human sight is a double-sided sword for visualization purposes: artificial representation of the environment of the simulator can be immersive and natural; however there are many opportunities for provoking nasty side-effects such as fuzziness and eye strain if not done correctly [111]. The main purpose of a visual representation in a simulator is to convince the users that what they are seeing is real. This is, in many times, not as related to graphical quality as to a good and plausible representation of the motion of the mechanisms. This is the field where multibody simulators excel since the output of this software are the real —or at least realistic— motion of the bodies in the world due to the user interaction or any other forces in the simulation.

Today, the majority of graphic displaying devices are aimed at the computation and rendering of computer-generated imagery. The devices needed for this activity are ubiquitous in the market, as they are used in many well-known and home-used tools, and thus their cost have been shrunken greatly. Screens, Graphics Processing Units (GPUs) and processors, are already present in a myriad of devices used in daily life. Computer-generated imagery has become one of the most important medium in human-machine interaction. Another emergent, promising field, is the denominated Augmented Reality [2]. This set of techniques combines computer-generated and real imagery in order that the user can enjoy automatic feature recognition, or data displaying directly over the sight space of the user. That way, virtual objects and real objects need not to be in separate environments, and effects from one of the *worlds* can be seen in the other.

The most used devices for image representation are screens, although there are a vast number of types depending on their use and the visualization requirements. Any of those types have been used in the implementations of numerous simulators. Here, two kinds of image representation devices are presented:

- Projection screens: light reflecting or transmitting surfaces onto which images are projected. The light beams forming the image are cast from a projector device placed in front or at the back of the screen. The screens are usually not expensive and can be set to cover large surfaces, even non-planar ones. The projection for non-planar surfaces usually requires special lenses or mirrors, and therefore needs an additional distortion correction pass. Quality of representation is not the strong point of this technology, although efforts are being made in order to incorporate LED technology into those devices.

- Emissive screens: those which, by means of light emission, are able to display the desired image onto their surface. Usually those devices have a greater contrast ratio and therefore better display quality, but the price for surface unit is high and are generally available only as flat surfaces. To overcome the last problem, groups of screens can be arranged in order to cover any surface. This technique is commonly named *power wall* [63].

Projection screens can be adapted to any place because of their non-rigid nature, and are the best solution for displaying large images to a wide audience. They are also used in immersive cabins like Computer Aided Virtual Environments (CAVEs) [22] and domes. Those two screen rigs allow the users to be *inside* the rendered world, covering the most or all their sight field. On the other hand, apart from the small contrast ratios, the placement of the projectors is often an issue. For CAVEs, is not possible to have them into the cubicle, so they have to be set outside of the screen, wasting space since no object can be stored between the projector and the screen. Projecting from the back of the screen also result in losing image contrast and definition. This effect can be minimized by using mirrors, although this affects the projection quality. The same is true for spherical domes, where a fish-eye lens equipped projector has to be placed in the best viewing position, the center. There has been also research about moving the projectors the farthest possible to the center of the dome using spherical mirrors, a technique which leverages the cost of the installation since it avoids the use of a fish-eye lens [13].

Emissive screens deliver better color quality, definition (since they experience none or very little light scattering) and resolution. Nevertheless, their cost is usually higher and require more units and controlling computers to cover the same amount of surface than projection screens. The reason is the limit in size of each panel, and the limited number of output ports of each controlling PC (typically two per card). As described previously, a big projection surface can be built on a array of screens called *power wall*. Synchronization issues can arise between the refresh timings of each monitor. On the other hand, there is no distortion to correct, since all the projected surfaces are flat; only a perspective offset must be trivially computed for every screen.

5.2.1. Real-time rendering

Computer graphics have come a long way since they started at the beginning of the computing revolution. The representation of the results of the computer calculations is almost as old as the computing itself. However, several algorithms and procedures could only be about to be used in real-time environments due to hardware limitations. Texture mapping was developed in the seventies, but it only become common in real-time systems twenty years ago. In the same way, program shaders were used for offline rendering much earlier than they started to run on popular and commonly available devices. Graphics hardware and software is continuously evolving to implement latest research. Fortunately, the massive popularity of mobile devices has aided to streamline and lower the cost of rendering devices. Chip manufacturers are able to deliver low cost, low power consumption graphic processors that are able to implement modern functionality like real-time 3D graphics featuring programmable pipelines, called *shaders*. The fact that 3D graphics acceleration is now ubiquitous aids greatly in the development of simulators in a large amount of heterogeneous devices and operating systems.

From the developer point of view, experience on graphics programming and implementation of common algorithms is embedded in popular Application Programming Interfaces (API), that abstract the programmer from that previously mentioned heterogeneous set of devices. Graphics APIs provide high-level interfaces in order to command the hardware to perform the most usual graphic duties. As said, in the latest decade, the introduction of real-time programmable pipelines allowed the programmer to customize almost all of the render stages which the data to be represented is going to visit. Thus, the programmer can implement new lighting models instead of relying on the provided ones, load any kind of binary data into the card and use it as an additional input for the graphics processing, etc. This chance of transferring arbitrary data from the computer to the GPU, and the vast parallel-orientation of these devices ended popularizing the GPU computing field, which is a discipline on its own.

Nowadays most used graphic APIs for real-time rendering are OpenGL and DirectX. The main advantage of the OpenGL API is its presence in almost any 3D accelerated hardware platform, and even on non-accelerated ones by means of software implementations as Mesa. Therefore, OpenGL has become the *de facto* cross-platform graphics API. DirectX is the API from Microsoft, and is very popular because the large market share of this company's operating systems among home computers and video-game consoles. Given the high quality of today's video-games graphic environments, this is a good showcase for the performance and capabilities that what this API is capable.

For embedded and mobile platforms, dedicated APIs have been developed. In the case of OpenGL, there is an OpenGL ES API aimed at embedded platforms, featuring a modern graphics pipeline for accelerated graphics rendering. This API is used in the most popular mobile platforms today, as Google's Android platform and the Apple devices.

There exist as well higher level software libraries that leverage the developer from performing common, repetitive tasks that a graphics program has to implement: loading 3D objects and textures from disk files, culling the scenery out of the user's sight space, creating graphics contexts and rendering surfaces. Those are common tasks that have to be carried in every graphics project. Open Source and commercial frameworks are available for helping the developer in that sense. These frameworks are based on the graphics APIs that access the rendering hardware. As Open Source frameworks examples, the Open Scene Graph and Ogre3D can be mentioned.

Open Scene Graph is a modular, cross-platform toolkit that can be used for any kind of visual representation of computer generated environments. It supports all the OpenGL versions, including OpenGL ES for mobile platforms (Android, iPhone). The main idea of the library is to implement the graph theory to the layout of the scene to be rendered. This graph structure can be used as a hierarchy that will speed up several common computations as visibility culling, as well as help in state-sorting the objects to be rendered, resulting in a performance increase.

Ogre3D is another graphics rendering library that focuses on modular and versatile design, by adding new functionality through a plug-in system. Almost any component of the library can be selected and replaced at run-time with the plug-in system. As a demonstration, and besides its cross-platform nature, Ogre3D can use OpenGL as well as DirectX as renderer back-ends. It has also a very flexible material system that can be based on the fixed pipeline or on user defined shaders.

On the commercial side, one of the most popular graphics toolkit is Unity3D, which not only implements rendering tools but a large set of authoring tools for creating interactive 3D applications. This library is used by companies developing commercial applications as well as hobbyist developers by means of its free of cost license. The advantage of this toolkits lies in the polished, versatile environment that the user can experience from the beginning to the end of the project.

In a real-time simulator, graphics rendering shows the user what is the current status of the whole system so further actions can be taken in order to accomplish the desired tasks. Typically, the main loop of the program would take a simulation step, and then command the graphics devices to display the scenery according the new computed position parameters. However the frequency of the simulation and the displaying algorithms are very different. While a simulation step can be run at hundredths or even thousandths of a second, graphic devices do not usually have refresh rates above 100Hz, being 60Hz the most common frame rate employed in simulation. The frequency needed for creating the motion illusion is far lower than the order of magnitude required for representing other phenomena. Thus, a feasible strategy must be found in order to couple and guide the communication between the different devices and algorithms.

Multibody dynamics and graphics representation synchronization

Here it is presented a time synchronization method for a multibody dynamics simulation and a graphics software and hardware visualization system. This solution relies in a *single threaded* scheme, avoiding the overhead due to the use of operating system thread synchronization primitives. At the same time, it is simple enough to be able to offer additional features as time line recording and time stretching.

Since the performance of the multibody algorithm varies depending on the specific state configuration at each instant, it is generally not possible to compute the fixed time slice that a simulation step will consume. This method performs the dynamic computations on demand, by calculating as a first step what is the time lag between the last computed state of the mechanical system and the current system clock value. Therefore, once the temporal offset between the simulation and the real-time cursor is known, the number of required integration steps are trivially obtained. Afterwards, those integration steps are performed successively. At this moment, there could still exist a time delay between the current multibody time mark and the real time. Nevertheless, since each integration time step is computed in less time than the time step itself, that offset usually decreases up to the point where there it is at most one time step delay between the real time and the simulator time.

At the end of every integration step batch, commands for the graphic representation of the state of the mechanism can be issued. Additionally, any other task involving queries for the state of user input devices or any other interaction channel can be also performed at this stage.

At first glance, a common first approach for intermixing the simulator computations and its graphical representation is to dedicate a unique *thread* of execution for each of those two tasks. At run time, the *thread* in charge of the graphic rendering would read available data from the *thread* carrying the multibody algorithm, and then issue graphic commands while simultaneously the computation *thread* performs more work.

When the displaying system does not impose a complexity level that would require to develop a very convoluted code for driving the rendering devices, the use of multi-threading techniques does not offer many advantages, if any. Parallel code shows its best performance when the tasks to be carried simultaneously are so independent that the synchronization between them is taken to the minimum possible level; otherwise, the possible performance gains are vanished by the overhead imposed by the cost of using the synchronization primitives of the operating system.

In this case, both threads would be continuously reading from and writing to the memory segment where the location values for the simulation elements are stored. In order to avoid that one thread could, for example, read those values while the other is updating them, and therefore render them in an inconsistent state, further actions must be taken. A operating system *lock* can be created, preventing one of the *threads* to read the memory segment while the other is using it. However, locking primitives impose a computational overhead for themselves, specially noticeable when they are used every short periods of time.

Another consideration to be taken into account is the aforementioned fact that the rendering is performed at low frequencies, when compared with the simulator pace. There is no need to command the graphics hardware to render a thousand frames per second if it is not capable or designed to work that way, nor the user is capable of seeing all of those rapid succession of images. High frequency display rates have their uses, however, specially when a human user is not involved, for example for robotics artificial vision.

As a last point, it should be also considered that on accelerated graphics hardware, a great part of the computations is done on the GPU, and thus in parallel with the tasks that the CPU could be carrying at the time. The bottlenecks of the GPU are many frequently related to the *slow* data transfer between those two devices. This is the reason because the trend in graphics hardware is to minimizing as much as possible that communication by uploading all data to the GPU: textures, three-dimensional meshes, even shader programs are uploaded once. The GPU can create images and data programmatically that it could also use without having to involve the CPU. Ideally, the CPU should behave as a command issuer device, and therefore, spending the shortest possible time dealing with graphic operations.

The single threaded synchronization method described in advance is not only designed to synchronize the multibody software steps with a real time clock, but it can be also used to support smaller time scales, featuring a *slow motion* effect that could even be useful in interactive simulators in order to perceive very rapid movements. Optionally, the results of the simulation can be stored in order to be able to return the simulation to a previous, past point in time, or for off-line reviewing purposes.

A displaying-computation synchronization loop has to distribute available computing time into two tasks: graphics rendering and multibody computing. The latter task includes the motion computation and its required input reads. Given the interactive nature of most of the simulators, it makes sense that in the loop, multibody computations are prioritized. For a time slice Δt, a well behaved multibody solution will compute its motion at a faster pace, $t_{mb} \leq \Delta t$, being t_{mb} the time spent on the computations. Thus, if a simulation is delayed with respect the system clock, it can recover its time position calling the multibody system repeatedly.

Note that the term *real-time* is not used here in the hard sense: it is not guaranteed that, for each loop, the computations are carried in the exact instant that the system clock points to. However, it is true that the lag between the simulation time and the system clock is kept very small most of the time, if noticeable for human-interactive purposes. Furthermore, when dealing with graphic systems, is also not possible to assure when a task is going to start or finish: the detached nature of those systems — for performance reasons — makes it very difficult to force them to perform such fixed schedules. In that case, the graphics system should be taken apart from the main loop, therefore hindering the rest of the real-time tasks. The communication between both subsystems could be done by means of a queue buffer, but this can create a new communication bottleneck, as described earlier.

The simplest synchronization loop is shown in Figure 5.2. There, the dynamic computations are performed, thus advancing the simulation time, until it reaches the system clock mark. With the information up to date, the graphical representation is dispatched.

A more complex and feature-rich synchronization loop is presented in advance. It allows to set the time scale of the simulation time, $s = t_{simulation}/t_{system}$, or even go back to previous instants, where $t_{simulation} > t_{system}$. As the simulation advances in fixed time steps, this control loop is implemented in terms of frames. Each frame holds all the necessary information for rendering the scene at that point. Usually they are composed of the transformation matrices for all the objects, plus some additional parameters. The overall flowchart is shown in Figure 5.3.

In the same way as it was done in the simple loop, there is a check that finds if there is simulation data available for the current instant. The corresponding frame is computed as

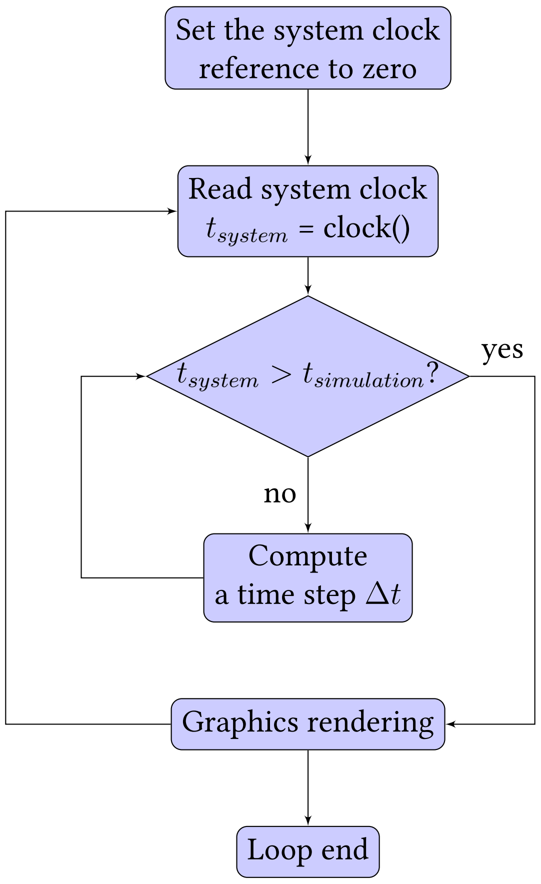

Set the system clock reference to zero
Read system clock t_{system} = clock()
$t_{system} > t_{simulation}$?
yes
no
Compute a time step Δt
Graphics rendering
Loop end

Figure 5.2: Simple synchronization loop

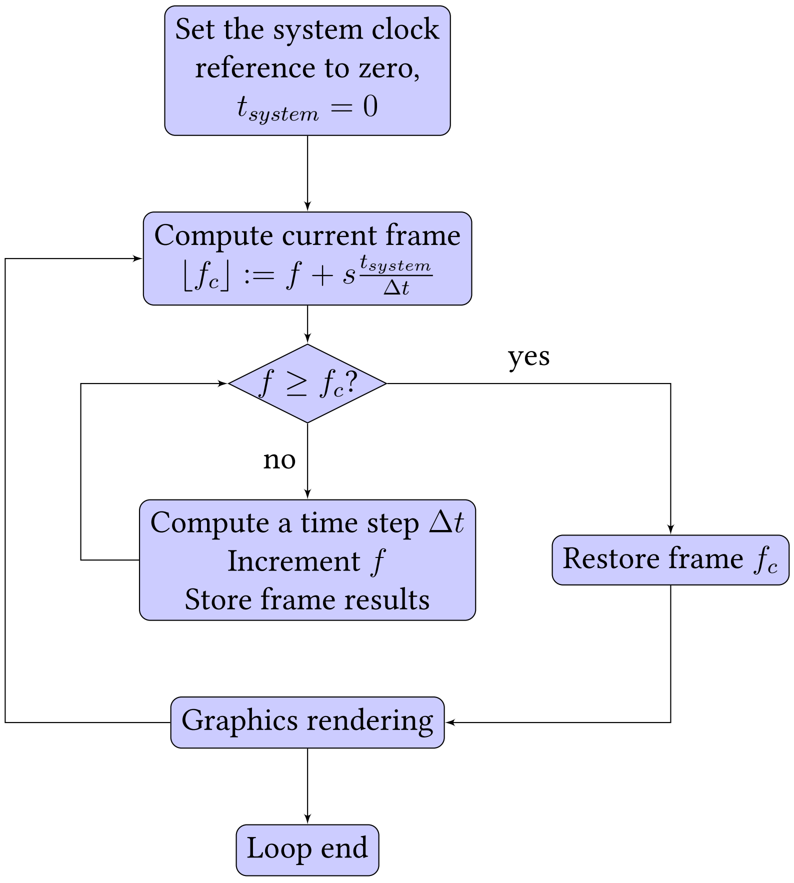

Set the system clock reference to zero, $t_{system} = 0$
Compute current frame $\lfloor f_c \rfloor := f + s\frac{t_{system}}{\Delta t}$
$f \geq f_c$?
yes
no
Compute a time step Δt
Increment f
Store frame results
Restore frame f_c
Graphics rendering
Loop end

Figure 5.3: Synchronization loop featuring time scaling.

$$\Delta f = s\frac{t_{system}}{\Delta t} \tag{5.1}$$

where f is the number of computed frames, and t_{system} is the elapsed time since the scale s was changed. The frame count is updated as an integer variable, $\lfloor f \rfloor := f + \Delta f$.

If $\Delta f \neq 0$, additional computing iterations must performed. At each of those iterations, the results of the simulation are stored for later reusing.

Each time the scale s is altered, the system clock has to be reset, $t_{system} = 0$.

5.2.2. Real-time rendering of reflective surfaces

Part of the immersive impact of a machinery simulator relies directly in the realism and fidelity of the virtual cabin offered to the user. There, not only the maneuvering controls and displays should have a similar feel as the real ones, but they also have to offer the same functionality. A simulator interface that does not impose the same operation sequences on the user can become a counterproductive experience for novices, because they are allowed or forced to perform unrealistic gestures. For real experienced users, the simulator as a certification or upgrading tool might become as well an unpleasant and frustrating experience.

In that sense, the correct representation of the auxiliary mirrors of the machine cabin can be a crucial part of the simulator. In the real machine, the mirrors allow operators to make quick glances at work zones initially out of their sight, therefore behaving as an additional source of information. Machinery maneuvers into limited working zones make them essential in order to finish successfully the planned tasks. Not replicating the effect of those devices into the simulator induces their users to perform unnatural gestures in order to compensate the lack of information: most of the users would, for example, turn the cabin back in order to check the placement of their machines and their distance to possible obstacles which can be inadvertently collided. As a result, the simulator users would be taught inefficient ways of solving problems that do not even exist in the real world.

The challenge for implementing mirror surfaces into a graphical displaying system is twofold. First, a correct representation of the mirrored scene from the point of view of the user must be computed. Second, the resulting image must be superimposed into the main graphical representation of the scenery.

When facing a planar, regular reflective surface, the incident rays of light are usually reflected in the same angle with respect to the normal of the surface, i.e. $\theta_i = \theta_r$ Most of the mirrors have this property, although some are modified in order to enlarge the reflection areas of interest. In the Figure 5.4, it is shown that reflected rays from the viewers' position, v, match the symmetric projection of the user's view over the mirror plane, v'. Therefore, a corresponding relative projection orientation must be computed from the current view. The matrix defining the transformation between the current and the reflected view can be derived over the applying of a series of simpler transformations:

$$\mathbf{M}_r = (\mathbf{M}_m \mathbf{M}_v)^{-1} \mathbf{S}_{-n} (\mathbf{M}_m \mathbf{M}_v) \tag{5.2}$$

The first transformation is used for changing from the viewing coordinate system to the mirror system. Being $\mathbf{M}_v$ the *modelview matrix*, and $\mathbf{M}_m$ the transformation matrix that places the reflector in the scene, the transformation $(\mathbf{M}_m \mathbf{M}_v)^{-1}$ passes from the first to the second coordinate system.

The next step is changing the orientation of the view with respect to the mirror plane in order to represent the scene in front of the mirror. To accomplish this, the transformation must invert the scene by means of scaling it along the

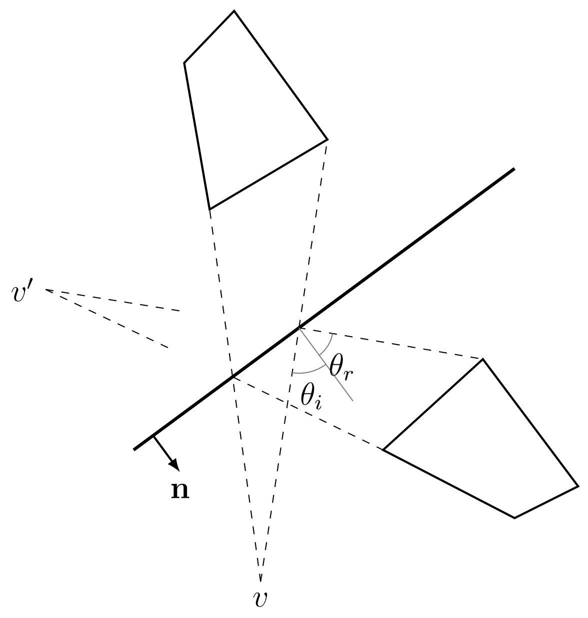


Figure 5.4: When the reflected angle is equal to the incident light angle on a mirror, the reflected image is equivalent to the result of rendering the image from a symmetrical point of view, v'.

normal axis of the surface. A unitary scaling about the axis makes the view point outside of the mirror. This is denoted as $\mathbf{S}_{-n}$ in the equation (5.2). The axis where the scaling is performed depends on the default coordinate systems used by the graphics framework and the one in which the mirror object is defined. In the case of the OpenGL graphics library, which defines x and y as the axis in the screen surface, and z as the axis pointing towards the screen,

$$\mathbf{S}_{-n} = \mathbf{S}_{-z} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5.3}$$

Finally, once the point of view is oriented to the other side of the mirror plane, the same relative position must be restored in order to replicate the reflected view. This is done by applying the inverse of the first transformation, $\mathbf{M}_m\mathbf{M}_v$.

Ultimately, the whole transformation process meaning is just the reflection of the scene along the local vector of the plane mirror.

As implementation details, the reflected image is rendered to a special buffer in the graphics system, so it can be later imposed over the main view. A technique must be used in order to establish which parts of the reflected image must be seen

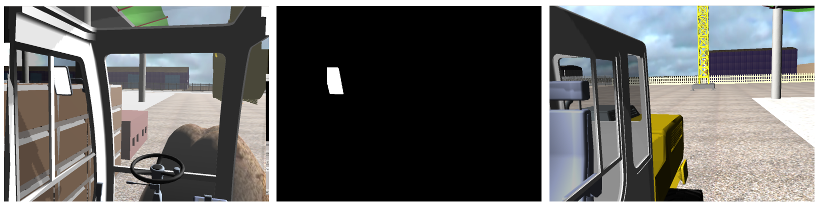


Figure 5.5: Pictures corresponding to the main view, the stencil mask and the mirror view.

because they correspond to the same area as the mirror graphical representation, and which must be hidden because they are laid out of it.

The most common aid that graphics hardware provides for this is the called *stencil buffer*. The *stencil buffer* is a multi-purpose memory buffer that can be used to store some information about the drawing process at the same time the graphics primitives are processed. The software can command to store certain values into the *stencil buffer* when drawing any object of interest. In the same way, the drawing of any object part can be conditioned to the existence of a certain value in the *stencil buffer* for the particular pixels that are about to be drawn. This is somewhat analogous to the *depth buffer*, where the objects' distance to the projection plane is stored for every pixel in the image. If, at a later stage, another object is sent to the hardware for drawing, the graphics system can know which pixels to draw and which discard, if they are placed further than the last primitive occupying the same place.

The stencil buffer allows the user to mark the pixels drawn by the execution of the rendering commands for a certain geometry. Therefore, a masking system can be implemented with this feature. For the mirror implementation, the stencil marking can be set only when the graphical representation of the mirror object is about to be done; later it will be deactivated as well. There is no appreciable effect in the color buffer, but the stencil will hold the area where the mirror surface was drawn (Figure 5.5).

A final rendering pass is completed afterwards in order to copy the reflection image into the scene. The stencil test is set to discard all pixels in the image whose stencil counterpart has not been marked in the process before. Therefore, only marked parts of the image corresponding to where the mirror object was drawn are effectively transferred (Figure 5.6).

5.2.3. Stereoscopic visualization

Currently most popular display systems are flat screens. Either as monitors, televisions or small hand held devices, they try to represent the computational results of the computers driving them in a human-friendly manner. This is specially

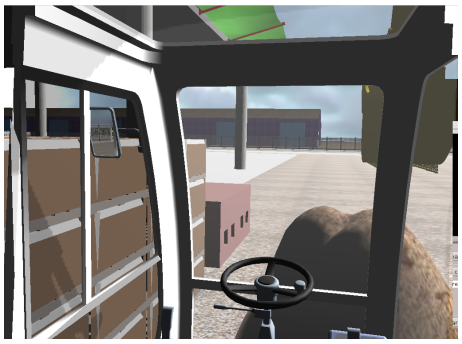


Figure 5.6: Final composited image.

important when designing simulators, since the graphical display device constitutes the most important feedback between the user and the machine. Thus, the representation system should aim to mimic the human sight and provide as much information and with the same quality as a person sees the real world. Immersive environments improve the interactivity of the simulator because the user will know better how to react to the stimuli presented. In fact, unrealistic representations can even cause adverse effects on the user such as *motion sickness* and discomfort. Human brain constantly processes any stimulus a person can perceive, and react as a consequence. Confusing, incoherent stimuli entails additional interpretation stress on the brain, leading to these undesirable effects.

Stereoscopic visualization is a technique that enhances the interactive degree of a simulation. On the other side, if not implemented correctly, it can be one of the sources of the motion sickness mentioned earlier. This visualization method consists in rendering two times each visualization frame, each from a slightly different point of view, as it would correspond to the images the user could perceive with both eyes. A planar perspective projection scales the dimension of the represented objects according to their distance to the point of view. This generates a three-dimensional effect, but at the same time it discards some of the position information: there exists an ambiguity between sizes and distances. A user cannot tell if an unfamiliar object is far away or if it is very small. The redundancy in the human sight overcomes this problem, since the offset between the objects in two different projections can be used to estimate their depth. Stereoscopic systems provide those two points of view as well in order to mimic what happens in the real life. The challenge is how to provide each of the users' eyes a correct display of what they should be seeing.

Simplest solutions merge the two images together into the projection, and later filter them for each eye (Figure 5.7). These methods are called *passive stereo*, since the filters used to separate eventually both images are not dynamic devices. In fact, they are usually surfaces made from materials that block one of the images

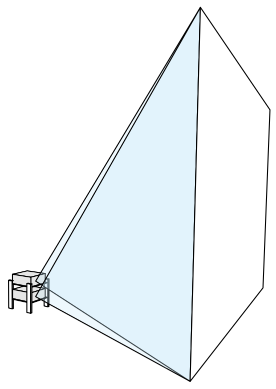


Figure 5.7: Stereo screen projection with two projector devices.

while transmitting the other. Anaglyhpic filters usually are made with chromatic filters that block one color component. They are usually employed in cyan-red pairs. Each one blocks the light image with its same component, and lets the other pass through. That way, eliminating a certain component from the image it can be made effectively *invisible* to one of the filters, while it can still be seen through the other. The disadvantage of this method is the alteration of the color for each image for the filtering: received images lack one of the components in the process.

Polarization filters work with the orientation of the light waves, leaving the chromatic quality intact. This system uses two projection devices, each of them projecting one of the images. Each light beam passes through a polarization filter which will orient it to be contained into a particular plane. If each beam is polarized into a different orientation, or even better in perpendicular ones, the user can receive both images at once. Each of the filters of the user blocks any light component not oriented in the same way of the filter, so each image can be again recovered from the mixed projection. The disadvantages of this method lie in that its performance varies if the user filters are not correctly aligned with the projectors'. Since the first are usually mounted into a pair of glasses that the user is wearing, any head tilting will cause a slight blocking of the image to be viewed, and the merging with the other. Furthermore, those filters affect the luminosity of the images, making the rendering darker. Using spherical polarization filters can alleviate the first problem, at the cost of having an even darker image as a result.

There exist other passive image separation methods, such as the *auto-stereoscopic* effect [99]. This was popularized with the Nintendo 3DS game console, which featured this technique. Here, the user is not required to wear any kind of glasses. The screen implements some barrier strips that block certain parts of the image from some points of view. That way, the discrimination between the two images can be carried at the cost of reduced screen resolution, since the display area is distributed evenly between both images. In addition, head movements could invert the stereoscopic effect if the eyes reach the other part of the barrier.

Active stereo systems allow to preserve all the image quality and allow the user to move freely at the cost of an increased implementation complexity. The projection device or display renders each pair of images in succession. The user wears a pair of glasses that are synchronized with the projection and blocks one eye while the other can see its assigned scene. Usually the glasses filter consists of liquid crystal panels that can be obscured in fractions of second. As the device receives the signal pulses from the display device, it blocks either one or the other panel. The eye-switching is so fast that the user does not perceive it, but at the same time each independent image can be seen through its panel. The cost of the special equipment required to implement this system is higher, since special graphic cards with synchronization ports and active LCD glasses are needed. It is expected to decay nevertheless, as this technology is progressively more used in home TVs and PCs.

Portable stereoscopic devices are implemented usually as *Head Mounted Displays* (HMDs). Two display panels are embedded into a pair of glasses or a helmet, in a disposition similar to a pair of binoculars. Each eye only sees its own display screen, maximizing the stereoscopic experience. The disadvantages of these type of devices consist in that usually the screens are very small for price, miniaturization and weight reasons. Even if the display channels are completely independent for each eye, the small image size and the narrow field of view can break apart the three-dimensional depth effect. The best selling point of this technology is not having a static screen, so the user does not have to stay always in front of it, and can move freely.

Moving freely into an environment with static screens can also be possible with the CAVE technology, which encloses the user into a cube and projects the corresponding scene into its walls. This technology allows as well the use of *active stereo* visualization systems, but has the downside of being only useful for a single user, as the projections in the walls are computed for this persons' location, and will be slightly incorrect for the rest of the users into the CAVE.

5.2.4. Hemispheric screen

Curved screens are good candidates for overcome some of the problems derived of the use of planar projection screens. These screens do not require that the user line of sight is perpendicular to them, as it is the case with planar screens without a head-tracking device. A curved surface projection can represent a correct display for all the orientations it covers; if a tracking system is not available, however, it only works for a fixed point of view. This is the most common case with machinery simulators, where the user is seated, and almost immobile.

In the same spirit as the mirror representation technique discussed earlier, a hemispheric screen makes possible to have a higher degree of immersion, because the field of vision of the user is enlarged to match almost the one in the real machine's cabin. Even with the aid of mirrors, a flat projection screen cannot take into account the peripheral vision effect of the human sight. Users would have to perform unnatural cabin movements in order to see what items are placed in the immediate surroundings of the machine.

The drawbacks of this method are twofold: first, as the vast majority of the graphics rendering system are based in a planar projection, the result image should be eventually corrected in order to compensate the perspective distortion in the projection stage. Second, in order to extend the image across all the projection surface, special lenses with wider field of view are required. These lenses can usually cover 180° or even more. In order to exploit all the available field of view that the lens provides and to have an homogeneous pixel density across the surface, usually the lens should be placed at the center of the enclosed surface volume. Users may intercept the projection when moving inside the volume.

For some configurations, more than one projector must be needed in order to cover all the surface. This usually results in overlapping areas at the image boundaries from several simultaneous projections. This side effect must be also corrected by crafting a luminosity map that dims the intensity of the overlapping areas.

Bourke [13] proposed the use of spherical mirrors in order to avoid the use of special lenses and to be able to use off-the-shelf projectors, therefore reducing costs. It is still necessary to perform distortion correction and luminosity compensation, although this technique can allow to move the projector out of the volume center.

As a last point, it should be noted that this type of projection permits to use any type of shape, and indeed it is very common the use of inflatable domes for astronomical visualization applications.

For implementing a hemispheric projection software, the initial steps are similar to those used to render the six views for the walls of a CAVE. This is necessary in order to compute the view of the scene from any possible orientation. Unfortunately, in this case, further processing must be performed with those images, instead of simply *sticking* them into the CAVE walls.

Since the screen is curved, for every point in its surface, the corresponding color point in one of the views must be fetched and drawn. The projection could be compute for every pixel, but reasonable approximations can be found as well. One is to approximate the surface by a polygonal mesh and to apply the image views as a texture onto it. Only the projections for the vertices of the mesh must be found, and the rest of the pixels are interpolated linearly by means of the texturing algorithm. The approximation precision can be adjusted by modifying the resolution of the projection mesh.

This kind of texture mapping can be accelerated by hardware most of the times. Usually graphics systems provide a *environment mapping* texturing mode. This mode is designed specifically for the rendering of reflection, refraction phenomena or any other rendering effect dependent on the normal of the drawn surface for each of its points.

The environment mapping requires as its inputs the six views corresponding to the projections of the scene into the sides of a cube. This is the origin of the *cube mapping* denomination. At the beginning of each rendering frame, the scene is rendered those six times and stored into six texture objects. All the possible views of the scene in any direction is present into those textures. In a different way to the other texturing modes, the texture coordinates are triplets (r, s, t) represent the ray defining the projection orientation for each vertex. Indeed, the triplets can be regarded as the components of the direction of the ray, $\mathbf{n} = (r, s, t)$

The output from the graphics card is a pre-distorted image of the environment, transformed in a way that will present a correct perspective to the viewer. In order to distort the image, a flat grid mesh is drawn onto the graphics' card buffer. The mesh is covered by a texture that agglutinates the different views for the *cube mapping* [54]. Graphics systems fill the rectangles of the grid with the information provided by the texture coordinates $\mathbf{n} = (r, s, t)$ specified at each of its vertices. Those values are obtained from the relative position of the observer from the projector lens.

The grid distributes the projection space homogeneously. Taking into account the extreme angle projection from the fish lens, the angles in a spherical coordinate system can be guessed. In this approach, a system capable of rendering into a $180° \times 135°$ sphere dome is demonstrated. The polar angle is distributed evenly along the spherical cap, $-90° \leq \theta \leq 90°$, while the azimuthal angle is distributed over a range $90° \leq \phi \leq -45°$. Thus, given a grid point located at the coordinates (x, y), the corresponding angle coordinates are shown in equations (5.4) and (5.5).

$$\theta = \frac{x}{w} 2\pi \tag{5.4}$$

$$\begin{cases} \phi = 2\frac{y}{h}\frac{\pi}{2}, & \text{if } y > 0 \\ \phi = -2\frac{y}{h}\frac{\pi}{4}, & \text{if } y < 0 \end{cases} \tag{5.5}$$

The spherical coordinates (θ, ϕ) determine the direction of the incident ray from the projector which passes through the grid vertex. An intersection point between the ray and the dome can be computed from its surface equations. First, the direction vector $\mathbf{n}$ can be computed from the spherical coordinates.

$$\mathbf{n} = \begin{bmatrix} \cos\phi\cos\theta \\ \cos\phi\sin\theta \\ \sin\phi \end{bmatrix} \tag{5.6}$$

A point $\mathbf{r}$ must be found so it lies both on the sphere with radius R and the ray passing through the projector lens location, $\mathbf{p}$. The solution of the problem is the ray parameter α which defines the point.

$$\mathbf{r} = \mathbf{p} + \alpha\mathbf{n} \tag{5.7}$$

$$\mathbf{r}^{\mathrm{T}}\mathbf{r} = R^2 \tag{5.8}$$

Substituting (5.7) into (5.8) yields

$$[\mathbf{p} + \alpha\mathbf{n}]^{\mathrm{T}} [\mathbf{p} + \alpha\mathbf{n}] = R^2 \tag{5.9}$$

$$\|\mathbf{p}\|^2 + \alpha^2\|\mathbf{n}\|^2 + 2\alpha\mathbf{p}^{\mathrm{T}}\mathbf{n} = R^2 \tag{5.10}$$

leading to a second degree equation

$$\alpha^2 + 2\mathbf{p}^{\mathrm{T}}\mathbf{n}\alpha + \|\mathbf{p}\|^2 - R^2 = 0 \tag{5.11}$$

whose solution is

$$\alpha = \frac{-2\mathbf{p}^{\mathrm{T}}\mathbf{n} \pm \sqrt{4(\mathbf{p}^{\mathrm{T}}\mathbf{n})^2 - 4(\|\mathbf{p}\|^2 - R^2)}}{2} \tag{5.12}$$

Since vector $\mathbf{n}$ is already pointing to the correct direction, the only real solution is the one for which $\alpha \geq 0$, thus resolving the ambiguity between the two possible solutions of the equation. It has been assumed that $\|\mathbf{n}\| = 1$. Since the position of the lens is always located over the z axis, its position vector can be simplified to be $\mathbf{p} = (0, 0, d)$, being d the distance from the center of the sphere to the projector. Therefore, equation(5.12) can be simplified.

$$\alpha = \frac{-2dn_z \pm \sqrt{4(dn_z)^2 - 4(d^2 - R^2)}}{2} = -dn_z \pm \sqrt{(n_z^2 - 1)d^2 + R^2} \tag{5.13}$$

In addition, since $d \geq 0$ and $0 \leq n_z^2 \leq 1$, further simplifications can be performed.

$$\alpha = \sqrt{(n_z^2 - 1)d^2 + R^2} - n_z d = \sqrt{R^2 - d^2\cos^2\phi} - d\sin\phi \tag{5.14}$$

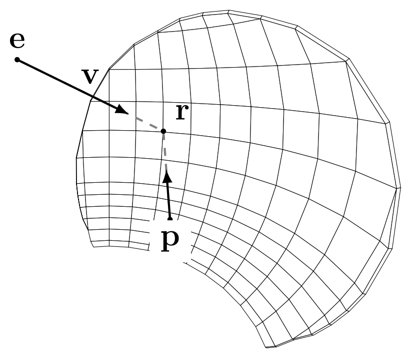


Figure 5.8: Computation of the viewing direction $\mathbf{v}$ from the eye point $\mathbf{e}$ for a point in the dome surface, $\mathbf{r}$.

Note that α only depends on the angle parameter ϕ, $\alpha = \alpha(\phi)$, and therefore it is constant along the latitude circles of the sphere.

Now that the position of the vertex drawn into the dome surface has been found through (5.7), it can be also calculated the orientation from vertex projection to the point of view of the observer, as shown in Figure 5.8. Observer's eye point $\mathbf{e}$ is the spatial point describing where the user will be seated. The orientation can be computed as

$$\mathbf{v} = \frac{\mathbf{r} - \mathbf{e}}{||\mathbf{r} - \mathbf{e}||} \tag{5.15}$$

The view vector $\mathbf{v}$ is defined as an unit vector, since this is what the graphics system expects. Its components can be interpreted as the texture coordinates for the environment mapping, $\mathbf{v} = (r, s, t)$.

Following this procedure, a set of texture coordinates can be found for each vertex in the grid to be drawn. It is only needed to know the position of the projector $\mathbf{p}$ and the observer, $\mathbf{e}$, and the radius R of the sphere. This mapping is immutable as long as the position of the viewer $\mathbf{e}$ does not change. If it is planned for the observer to move, the texture coordinates must be recomputed each time a new video frame is rendered. A trade-off between performance and visual quality can be established varying the resolution of the grid. In the same aim for performance improvement, the computation of the texture coordinates can be offloaded to the GPU by performing the computations into a shader instead of into the CPU. Note, however, that the mapping is independent from the observer's *orientation*, thus being a very convenient solution for machinery-simulator displaying systems where the user almost does not translate his head, but only rotates it.

5.3. Audio output

Although the human sight is the most developed sense, simulators can benefit from enhanced interaction environments if sound is added to the simulation. Sounds can be used as signaling events that warn the users about actions that can be happening out of their sight, happening contacts or impacts, etc.

Sound processor systems are found in almost any computing device, and today it is available a large amount of models offering surround sound emission. Software is also available in order to take advantage of the hardware. From stereo sound to more complex sets as 5.1 sound systems, audio APIs as OpenAL allow developers to program systems in a cross-platform way. Currently there exist four implementations of the API, being the original *OpenAL* from *Creative Labs* [72] and *OpenAL Soft* [91] the most popular.

Different implementations can be enhanced with a extension method which allows the software to query for new features and use them if available. This method was heavily inspired in the OpenGL extension mechanism.

Nowadays sound generation methods are mainly sample-based, instead of synthesized. Nevertheless, current computing systems' power makes affordable the generation of sound waves in real time. In fact, sampled sounds are usually transformed in several ways to add additional effects such as reverberation, volume distance attenuation, pitch change, etc.

APIs as OpenAL facilitate the implementation of spatial sound effects using the concepts of sound sources and listeners. Sources are assigned a sound sample and position. The listener has a defined position and orientation, and thus it can be computed what is the intensity of each sample in every loudspeaker in the sound system. Sources and listeners can also specify their velocity, so an additional pitch adjusting can be made based on their relative speed in order to implement the Doppler effect.

The operation mode for the OpenAL library is somewhat similar to its graphic counterpart, OpenGL. Indeed, both share the *state machine* model that tries to isolate the program from the hardware abstraction that performs the work. The hardware abstraction holds all the data about the desired sound generation, and it performs its commands in parallel with the CPU. From this point of view, the CPU acts only as a command dispatcher that issues a list of orders to the hardware abstraction queue, and it is later free to perform any other tasks while the sound subsystem finishes the commands. This is even more important for sound generation, as the order of magnitude of a sound sample is usually much bigger than the typical loop cycle of an interactive simulator. Sound samples can last several seconds, while the simulation loop can be executed in less than a 1/100 of a second.

The larger the sound buffers are, the more work that the CPU can defer to the sound system. Unfortunately, apart from the memory size limit of the latter, there exists other practical limitation for the maximum buffer sample size: sound latency. Sound latency can be defined as the amount of time between the detection of an event and the moment where the corresponding sound is actually played. For sounds that are being played in a continuous, looped form, the latency effect can be noticeable. If the sound stream must change any of its parameters while being played — the revving up of an engine is a good example — and the buffer is too large, it can take a noticeable amount of time since the user acts on the controls and the emitted sound is altered. The reason is that the sound system must end playing the current buffer before attending new commands in the queue. Depending on the specific simulator characteristics, a compromise solution for the buffer sound sizes must be found in order no to charge the CPU unnecessarily, while retaining a good response time.

As a first step when using OpenAL, a *context* must be created. A *context* is a software object which represents the current state of a hardware sound system such as a sound card, and it is use hereinafter for referring to such device. In addition, creating a context prepares the device for accepting new commands and resets its state to known, neutral values.

Each sound context can have only one listener device, which represents a person hearing the sounds. Parameters such as listener's position and orientation can be set so the sound system can alter emitted sounds accordingly as the listener moves.

However, a context can have, and usually it does, many different sound sources. A sound source encapsulates a sound sample as well as its position, gain, velocity, stereo or mono type... Hence, the OpenAL can compute the transformation of a sound source given the relative positions and velocities from the listener's perspective. For continuous mode, such as engine sounds, the `AL_LOOPING` option commands OpenAL to queue again a sound immediately after it has been finished playing.

Sources are created with the `alGenSources()` function and its parameters are set with `alSourcei()` and `alSourcef()`.

At each time instant, the sound subsystem transforms and joins all the sound sources into the sound channels for the speakers. The process of joining different sound sources into a specific channel is called *mixing*. Each channel corresponds to a speaker, so it is possible to have spatial sound using several speakers. For example, if a sound source is created and assigned to a position to the left of the listener, OpenAL will set a higher gain for the left sound channel in a stereo setup. More sophisticated speaker setups, such as 5.1 and 7.1, can deliver a more precise sound positioning, since they additionally have front and rear channels.

When it is desired to play a sound source, — as seen, it is queued and then mixed — the `alSourcePlay()` can be used. For continuous playing sounds, the counterpart `alSourceStop()` does exist. The latter can be also used to stop any sound source before its finish time.

The parameters for a sound source can be set any time, even when its sound buffer data is being played. Common real-time adjustments are their gain or their pitch. For the revving sound of an engine, its pitch — frequency adjusting — can be modified slightly in order to match the user's actions over the throttle pedal. Pitch shifting is not the most correct way of simulating this effect, but unfortunately, more complex methods are not effective for this specific case. Time stretching — changing the playing speed of a sound sample without altering its pitch — algorithms as WSOLA [110] rely on specific sound patterns that are usually found on speech and music audio files. For other types of sound samples, for example engine sounds, where no noticeable clear patterns are found, those algorithms cannot perform the task. Furthermore, they typically require a large window for its data processing, reaching sizes of 100ms. Those sizes can start to be noticeable in interactive programs.

5.4. Simulation Events: addition and removal of bodies in real time

Performance of a simulator is related to the size of the problem it is required to solve. Often it is desirable to run the simulator in a vast spatial domain, or the mechanism itself is very large when compared to the computational power available. Here will be shown a method that changes dynamically the size of the problem to be solved in order to be able to simulate scenarios that could be too cumbersome for the machine or machines where the software is running.

Usually, not all the bodies in a simulation are mobile, or it could be that the motion of some bodies can be uninteresting during most of the time. That is the case for scenery objects which could be potentially colliding against the mechanical systems being simulated. The goal is to consider into the simulation the lesser number of objects possible, thus keeping the size of the problem at a minimum. On the other hand, there are additional computational costs derived from changing the size of the multibody system: some long-lived, expensive to compute terms in the system must be recalculated and reordered if a body is added to or removed from the problem. Therefore, a sound strategy is needed to minimize the overhead of this optimization.

Testing which non-simulated objects should enter the simulation is performed by checking the positions of moving mechanisms. The bounding volume of a mechanism can be used as a guide for potential collisions; if a pair of bounding volumes intersect, they can potentially collide in the following time instants.

Hence, the inactive object should be added to the list of simulated bodies.

Removing bodies from the simulation can be done in the same way, by checking if their collision with other mechanisms is not possible because they are moving away from the other. Additional tests must be performed to check if the candidate object has finished moving; otherwise it could not be removed until it is still.

Bounding volumes can be modified in different ways in order to make the collision prediction more responsive and accurate. Enlarging the volumes according to the velocity vector of the mechanical system will consider mainly objects that are approaching at higher speeds. A simpler method consisting in increasing the size of the volume by a homogeneous scale can also detect approaching bodies earlier, but at the cost of increasing the number of considered bodies.

The overhead on the multibody reconfiguration is caused by the need of modifying the coordinate, constraint and forces arrays, in order to eliminate old bodies or to incorporate new ones. In fact, this change of configuration is analogous to the initial creation of the multibody system. If sparse matrix techniques are used, the symbolic factorization of the system matrix must be recomputed, since its topology will be different. New initial positions and velocities must be calculated to continue the simulation from the current instant of time.

5.4.1. Multibody addition and removal of bodies with natural coordinates

As an example, the process of addition and removal of bodies at run-time is shown for the specific case of the natural coordinates. Natural coordinates are mainly focused in characterizing a mechanism through points and vectors located at its joints instead of the global position of the bodies conforming it. A mechanism's design begins with the joints' specification, and then each body is defined with the existent points and vectors needed for the joints definition. Additional coordinate definitions could be needed in order to complete a body definition, for example, if its orientation is not completely stated by the joints.

For example, a three-dimensional pendulum can be defined by two points: one for the spherical joint and the other for the extreme of the bar. Nevertheless, in order to take into account the roll of the bar over its own axis, two additional, perpendicular vectors can be used in order to model the body. In a strict sense, only one vector could be needed, but doing so would lead to a non-constant mass matrix formulation. Therefore, increasing the number of variables leads to easier to handle systems at the cost of building slightly larger systems.

Another fact about this modeling method is that the characterization of a body is not unique. Any combination of points and vectors can be used for defining a body, as long as it defines a valid coordinate frame.

Body definitions using one point and three vectors are the most intuitive. However, this formulation does not impose that the point should represent the center of mass of the body, neither that the three vectors are orthogonal. The total number of coordinates per body is 12. Additional constraints must be used in order to preserve the rigid body state, narrowing the possible degrees of freedom to 6. Six constraints are enforced: three of them preserve the unitary length of each vector, while the rest preserve the relative orientation between those vectors. Those latter constraints can be dot product constraints, and they enforce that the dot product between any two vectors remains constant during all the simulation, that is, $\mathbf{uv} = (u_x v_x + u_y v_y + u_z v_z) = \cos\alpha = C$, being $\mathbf{u}$ and $\mathbf{v}$ two of the vectors, and α the angle between them.

Further point and vector combinations are used depending on the mechanism definition. A rigid body can also be defined by two points and two vectors: this time one unitary vector constraint is replaced by a distance constraint in order to impose the constant distance between the points. This pair of points can be assimilated to a non unitary length vector, and hence the orientation constraints can be modified in order to preserve their relative orientations:

$$(\mathbf{p_1} - \mathbf{p_2})^{\mathrm{T}}\mathbf{v} = ((\mathbf{p_1} - \mathbf{p_2})_x v_x + (\mathbf{p_1} - \mathbf{p_2})_y v_y + (\mathbf{p_1} - \mathbf{p_2})_z v_z) = \cos\alpha = C \tag{5.16}$$

Additional combination sets of points and vectors include defining a body by three points and only one vector, or just by four points. As shown in the previous cases, pairs of points can be used to define directions in the space in the same way that a vector could be used.

Many times, additional points and vectors apart from the minimal set required to define a rigid body are desired. This can be useful in order to create attachment links, or to define axes for other bodies. Since a body definition is equivalent to the rigid frame it represents, further points and vectors can be defined in terms of that coordinate frame. For example, for a given point $\mathbf{p}'$, a constraint can be imposed as

$$\mathbf{p}' - (\mathbf{p} + a\mathbf{u} + b\mathbf{v} + c\mathbf{w}) = \mathbf{0} \tag{5.17}$$

where $\mathbf{p}, \mathbf{u}, \mathbf{v}, \mathbf{w}$ are the point and vectors that define the body, and a, b, c the local coordinates of point $\mathbf{p}'$ in that frame.

A system capable of altering the mechanism definition at run-time requires additional implementation complexity. Apart from the overall system structures, as the coordinate vector or the mass matrix, there are some independent entities that have to be modified and updated. The list of bodies, body forces or constraints are examples of those entities. Instructions for updating those lists are given below.

The reason for the existence of a body data type lies in the interest of storing all the information relative to a certain body at any time. This body data type can be also used as a handle reference for attaching new force models to it, or for requesting any relevant information, such as the transformation matrix that defines its position and orientation at any time. For the purposes of removing the body, this structure holds the indices of the points and vectors used for its modeling, so it is clear which variables should be removed as well.

However, as noted with natural coordinates, any coordinate can be shared by two or more bodies simultaneously. Extra care must be ensured in order to make sure a variable is not currently in use before it is removed. For that purpose, a *coordinate user list* is created at start time: for every point or vector coordinate, a variable is incremented any time a new body is defined using that coordinate. At removal time, that variable is decremented for each of the coordinates of the selected body. If any of the counters for any of those coordinates would reach zero, that would indicate that the variable is not used anymore, and therefore it could be removed from the coordinate vector safely. This technique is called *reference counting* and is commonly used in Computer Science to keep track of used resources and for being able to detect when they can be released.

The coordinate vector can be therefore shrunk by removing the *holes* that the unused coordinates left behind: still in-use coordinates can be written into those new free positions, thereby compacting the vector. The formulation does not make any distinction between types of coordinates, being them points, vectors, angles or distances. In order to still be able to pack them into only one array, an extra indirection level is necessary: any coordinate is stored side by side in the array, but special index vectors for every type of coordinate are created in order to track where a specific coordinate is stored. This is shown in Figure 5.9. Vectors `ip`, `iv`, `ia` and `is` hold the real position of a point, vector, angle or distance variable in the `q` coordinate array. Therefore `iv(i)` would return the element in q that holds the i^{th} vector. This is a powerful concept, but nevertheless requires that the index arrays are also updated when the coordinate array needs it, thus preserving the coherence of the information.

Once all the coordinates related to a body are found and checked for the need of their removal, its associated constraint objects must be removed as well. Constraints are also encapsulated as first-class objects, since it allows to simplify the multibody code by the use of a generic concept. As a constraint is an arbitrary expression of the type $\phi(\mathbf{q}, \dot{\mathbf{q}}) = 0$, a generic, conceptual system must be devised to be able to perform high level tasks upon them (evaluate, differentiate, know which coordinates they use...) Each constraint object type represents a family sharing a common expression form. For example, *type 1* constraints represent

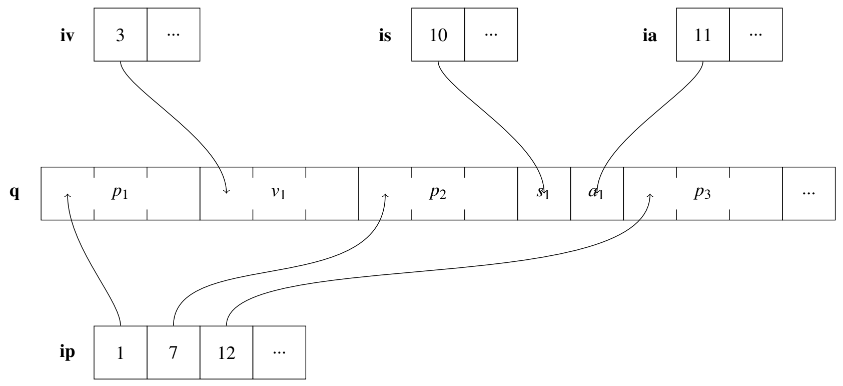


Figure 5.9: Layout of the coordinate vector $\mathbf{q}$ and the index arrays $\mathbf{ip}$, $\mathbf{iv}$, $\mathbf{is}$ and $\mathbf{ia}$.

the normal length condition of a vector, that is, $\mathbf{n}^{\mathrm{T}}\mathbf{n} = 0$. This specific kind of object holds the index of the desired normal vector. When created, it is added to the list of *type 1* constraints. At residual computation time, or when the Jacobian matrix is needed, all the constraint lists are queried for existing objects, and if they exist, they are used to perform the desired operation. Therefore, in order to remove completely a body from the simulation, all its constraint objects must be removed as well from each constraint list.

A body object does not hold explicit information about which constraints are used for its definition. However, attending to its modeling type and its coordinate indices, this information can be computed. As previously seen, when a vector is used for modeling, a normalization constraint is used. Searching into the normalization constraint list for the index of that vector, that constraint can be located and later removed. Since rigid body constraints are not shared, there is no extra bookkeeping to be done, and following constraints of the same type can advance one position in the list for occupying that new hole.

In a very similar case, force models are also encapsulated into objects. Each object type implements a different force model (a spring force model, a tire model, a damper...) but present a common interface, so they can compute their influence into the external force vector $\mathbf{Q}$ when the multibody system needs it. This time, the force objects do hold the bodies that they are affecting, so they can be removed without further computations from the force types' lists.

The list of degrees of freedom of the system is composed by all the coordinates that describe all its possible changes in position. After taking out any body from the simulation, all the now unused coordinates must be removed as well from that list.

After the removal stage, overall system terms must be recomputed to reflect the changes. Given equation(2.10) describing an ALI-3 multibody formulation, the removal or addition of any body into the system requires reassembling matrices $\mathbf{M}$, $\mathbf{\Phi_q}$, ${\mathbf{\Phi_q}}^{\mathrm{T}}\alpha\mathbf{\Phi_q}$ and $\mathbf{Q}$. Tangent matrix $\mathbf{T} = \mathbf{T}(\mathbf{M}, \mathbf{\Phi_q}, \mathbf{Q})$ has to be also symbolically factored.

Those procedures will typically consume a bigger time slice than the actual time step that it is being computed, delaying the simulation if bodies are entering or leaving the simulation frequently in a short amount of time. The overhead should be amortized by strictly checking if a body is eligible for changing its status; otherwise, this optimization can lead to poorer performance than the original one of the simulator.

Bodies are not the only type of entities that can be added or removed during the simulation. Modifying constraints at run time can provide meaningful results and a more flexible simulation environment. For example, constraints can be removed to detach parts of mechanisms, in order to represent machine damage. Other example case could be a machine assembling simulation where explicit joint constraints are created in real time as the user mounts the parts together.

5.5. Network process synchronization

Often, real-time simulators' complexity is too high to be carried on one single computing system. Other designs can fit best if each part of the simulator is isolated, typically in different machines. This can be advisable if there exist reasons such as the need to use exotic hardware that cannot be directly controllable by the main simulation system, or having a main computer that can start, stop and monitor several simulator seats. This scheme exploits the transfer performance of local area networks (LAN), which can be used for system communications in real time.

One popular layout is the separation of the simulation core from the graphics display system, if any or both of those systems has a high degree of sophistication. Simulator graphic systems can drive many displays at once, in set ups as *power walls*, CAVEs, etc. Again, often a single computer is not enough to provide all the required graphical outputs or processing power. The task needs to be divided into several different computer boxes that can handle each bit. Communication is implemented through a network layer where the information needed for the display of the scene is broadcast to all the computers. For mechanical simulators, this can be accomplished by sending the position of each of the parts and entities being displayed, plus some additional parameters in case special effects are needed, such as deformations, text messages, time-based effects...

As noted, the information packets for each instant of time only comprise mutable, dynamic information. Common data such as geometry assets is either stored beforehand in all the machines or found in shared accessible network drives.

Monitoring computers, controlling several simulator seats, can also exploit the available connectivity through the use of networking. A two-way communication process can be established between each simulator and the monitor system. Commands are issued to the simulators in order for them to perform any desired actions. However, a command protocol known to both systems is needed. Usually it consists on a list of numerical codes or command strings.

Computer networking is an advanced topic. Current networking systems exist as a stack of protocols or communication layers [100]. Each layer of the stack deals with different transmission tasks. Higher-level protocols are friendlier to user code, while lower-level layers have more to do with the hardware implementation of the link. Therefore, all the details of a specific communication are isolated from the parts of the stack that are not needed. This allows, for example, the lower-level protocols to be independent from the higher-level message to be transmitted. Also, higher-level protocols can successfully work without depending on the particular lower-level communication details. The most popular high-level protocols are UDP and TCP. They are both implemented over the IP protocol.

UDP is a *connection-less* protocol, meaning that can be used to send single data packets to one or more recipients in an isolated way, without having to establish any previous communication steps. This protocol does not ensure that each packet will be received, nor that the packet reception order will match the sending order. If, for application purposes, it is not admissible any packet dropping, or if it is crucial to preserve its receiving order, additional checks must be performed between both peers, re-sending the data should it be necessary.

TCP is a connection oriented protocol. Unlike UDP, data is not sent directly in packets by the user, but instead, a channel or stream of communication between the computers is established beforehand. The stream enables communication in both ways, and ensures that the data packets will reach the other point and in the same order as it was sent. Those features impose an added overhead over *connection-less* protocols such as UDP. On the other hand, implementing the features that TCP provides over UDP can be inefficient and it is not in any way a trivial task. The decision whether to use one or the other should be taken depending on the application requirements and design, in a case-by-case basis.

5.5.1. Serialization Libraries

When dealing with the least demanding cases, the design of a network packet format is trivial. If the elements of the data to be transmitted do not vary in nature or size during the simulation, a fixed ordering can be established, in which the transmitter can simply write them. The receiver does not have to do any extra processing after receiving the packets, since the location of every data item is already known. After storing received data into a memory buffer, any specific piece of data can be accessed directly within it. Some parts of the buffer can be ignored if they are not needed, just by reading only the blocks that hold important information.

However, when transmitted data has a more complex nature, where each transmitted packet can hold more convoluted data, as variable length character strings, linked or nested data structures, pointers or references to other chunks of data… a generic and systematic approach is needed. It is a typical scenario for simulations where objects can show up or disappear, different events are produced at any time, or very lengthy data types have to be transmitted from time to time. A first approach could consist on writing an initial index for the transmitted data where it is recorded which kind of items and how many of them are present in the buffer. For example, an event could be stored as numeric code for identification, and a string of text holding a description of the event. However, the string of text is also a mutable object, since it varies its size depending on the length of the message. The string should be encoded, in turn, trough the use of a number storing the length and then, the text data. Therefore, if one wants to skip a certain event, a computation must be carried for knowing how many bytes the next chunk of information is from the current position.

It is clear from this example that encoding that kind of information into the network stream can become a cumbersome and error prone activity. Hopefully, there exist already software solutions to address this problem. This process is called *serialization*. The name resembles the task of flattening a complex, multi-level data structure into a linear sequence of data elements, ready to be transmitted through the network or saved to disk. Serialization libraries perform *instrospection* duties. This means that they are able to identify the relationship between the elements of the data structures, and substitute them by references, should it be necessary. Thus, the programmer can forget about low level transmission details and just mark the pieces of data to be transmitted.

The reception process is also simplified. Once the data arrives, the introspection system is able to know which items — and their quantity — where present in the stream. The programmer can query for a specific item in the stream or load the structured information into memory.

Since serialization, or structuring data present in memory into any other medium, for storage or transmission purposes is a recurrent problem, many solutions have been developed. Nevertheless, it must be noted that those solutions can differ depending on the exact intended usage. Although, for example, XML language is regarded as one of the most versatile ways of encoding data in a hierarchical way, concerns about bandwidth usage — XML is a verbose protocol — stimulated the creation of new solutions for scenarios where bandwidth and computing restrictions were limiting factors.

Google Protocol Buffers

Some of those solutions are specifically aimed at reducing the overhead of the serialization process in bandwidth and processing costs. An example is Google's *Protocol Buffers* [51]. It is a very flexible system, where the user defines the data structures in a specific language. Its syntax resembles other general purpose languages such as C++ or Java. There, the user states which data structures are going to be transmitted, and which elements do form part of them. Those structures can be nested, and furthermore, each element can be considered as *required* or *optional.* Therefore, the sender can choose not to send some data blocks if it considers that they are not needed.

After creating the file containing the structures definition, the user has to process it with a provided scanner program which will, in turn, translate that syntax into the language of choice. Currently, C++, Java and Python are supported. The scanner will output the source files describing the data, in addition to some boilerplate code for managing them.

This is, however, a very intrusive method. The programmer is not even able to directly write the classes that are needed. The problem is magnified if the serialization feature is going to be added to an existing program, which can be very difficult to modify in order to fit those requirements. It is advised to use the serialization structures only for data transmitting purposes, even if it means having to duplicate the data already stored in other places in the program.

As an example, a scheme for transmitting the position of a body is presented in Listing 5.1. A mechanism's spatial configuration is defined by chaining the definitions of all the entities that conform the configuration. The *message* is the information unit that will be sent through the preferred communication channel or storage method. In a hierarchical manner, additional items can be defined inside a message declaration. In this case, the *mechanism* is composed by several *bodies*, which in turn have both *position* and *orientation* attributes. The *position* item holds the coordinates (x, y, z) of the frame of reference of the body, while the *orientation* item stores any four component parametrization defining the rotation of that frame: Euler parameters (e_0, e_1, e_2, e_3), quaternions (q_x, q_y, q_z, q_w), or axis-angle (n_x, n_y, n_z, θ) representations are adequate for this usage.

Listing 5.1: Mechanism scheme for transmission with Protocol Buffers.

```
message mechanism
{
  message body
  {
    message point
    {
      repeated double coordinate = 1 [packed = true];
    }

    message orientation_parameters
    {
      repeated double orientation = 1 [packed = true];
    }

    required point position = 1;
    required orientation_parameters orientation = 2;
  }

  repeated body part = 1;
}
```

The `repeated` keyword defines an array of aggregated, identical items. Its size is not explicitly defined, and can be later queried by the message's receiver. `position` and `orientation` array sizes are always expected to be 3 and 4 respectively, but `part` array length can vary depending on the addition and removal of bodies at real-time, as discussed on section §5.4.

The `body` definition could also be augmented with a `string` field for storing the name of the graphical file object that represents each part.

Boost::Serialization

Boost::Serialization [12] is a less intrusive serialization library. It has two modes of operation: in the first, the user writes a serialization function into the definition of the data class. That code states how to serialize the data by choosing its important data members. Each data member is sent to an *archive* object that will in turn call the serialization function for that data member.

The second mode is designed for the case where an existing data structure in the program cannot be modified in any way to include the serialization mechanism. In that case, an external function must be provided in order to pass the data of the object to the archive object. Unfortunately, that data must be accessible from functions not belonging to the structure, so it cannot be used to store its private members.

A example showing the use of the library is found in Listing 5.2. Basic data types for storing the position and orientation for each body are defined. A `body` type holds both together, and finally, a `mechanism` is declared as a collection of bodies. The serialization for a data type consists on passing each piece of data it holds to the archive object, `ar`. Thanks to the overloaded operator `&`, the same function `serialize` is used to write data as well as to read it back.

The serialization process needs to be called on the `mechanism` object. This process will automatically retrieve all the members in the hierarchy — bodies, points, parameters...—, by chaining function calls.

Listing 5.2: A reference multibody serialization

```
class point
{
  public:
  double p[3];

  private:
  friend class boost::serialization::access;
  template<class Archive>
  void serialize(Archive & ar, const unsigned int version)
  {
    for(int i = 0; i < 3; i++)
    ar & p[i];
  }
};

class orientation_parameters
{
  public:
  double o[4];

  private:
  friend class boost::serialization::access;
  template<class Archive>
  void serialize(Archive & ar, const unsigned int version)
  {
    for(int i = 0; i < 4; i++)
    ar & o[i];
  }
};

class body
{
  public:
  point p;
  orientation_parameters o;

  private:
  friend class boost::serialization::access;
  template<class Archive>
  void serialize(Archive & ar, const unsigned int version)
  {
    ar & p;
    ar & o;
  }
};

class mechanism
{
  public:
  std::vector<body> b;

  private:
  friend class boost::serialization::access;
  template<class Archive>
  void serialize(Archive & ar, const unsigned int version)
  {
    ar & b;
  }
};
```

Chapter 6

Implementations

In this chapter, an example of a real-life implementation of a simulator is presented. This concrete system illustrate how the techniques described in the last chapters can be combined in order to get a meaningful product.

A mechanical system simulator can have several purposes. It can boost the design and prototyping phases, showing its possible defects at an earlier stage, while minimizing the number of real prototypes built to test or it can also serve as a training tool, because it can mimic the real behavior of the machine with a high degree of fidelity.

The proposed system is a excavator simulator. Its main purpose is to be used as a training system for this kind of machinery. The challenge of using these machines is to master the use of all the available controls.

Hydraulic excavators are among the most versatile earth-moving equipment: these machines are used in civil engineering, hydraulic engineering, grading and landscaping, pipeline construction and mining. Their primary functions are digging, material handling and ground leveling. To execute these operations, the excavator operator actuates the machine controls (joysticks, pedals and switches) in an organized form to achieve the desired machine motion; the actuation of these controls is a complex and not intuitive task, and therefore it requires long and costly training periods [106].

In fact, just driving the excavator over the terrain is a fraction of the required skills for becoming a operator, although getting used to the size of the machine and avoiding obstacles can be a difficult task.

The simulator can aid to lessen the needed real-machine hours to a minimum, while at the same time providing a safe environment for starters. Simulators have been used for years in aeronautics for this very purpose.

The qualification of the excavator operator is not only required to operate the machine properly, but it has also a significant impact in productivity and safety. Bucket loading highly depends on the operator digging style, as demonstrated by Hall [58]; hence, skilled operators can finish the task in less time, with important cost savings, and the same applies to material handling and ground leveling operations. As far as safety is concerned, hydraulic excavators may be involved in two types of accidents. The first one is underground facility damage under excavator loading, in particular pipeline and power or communication cables. For example, it is documented that digging operations cause around 50% of the failures in onshore oil and natural gas transmission pipelines, see e.g. Brooker [15]. The costs derived from supply interruption, facility repair and time delays in the excavation are very high, and these accidents may also involve fire or explosion with severe possible consequences in terms of loss of life, injury, property and environmental damage. Since in most digging operations the excavator bucket is out of sight to the operator, it is difficult to prevent such accidents when the exact position of the underground facility is unknown. However, experienced operators can interpret subtle changes in the excavator movements to immediately detect a collision between the bucket and an underground facility, and stop the operation before causing severe damage.

A second type of accident is excavator fall or rollover due to loss of stability (Figure 6.1). Hydraulic excavators, especially those used in civil engineering, usually operate on irregular terrains with poor accessibility. It is a common practice to use the excavator arm as an additional support or impulsion element, as shown in Figure 6.3, to avoid obstacles or to maneuver in small working zones. Skilled operators are needed to execute these maneuvers without endangering stability. Falls and rollovers can also happen when lifting heavy loads on uneven terrains, with severe risk for the lives of the operator and nearby personnel.

Sales of hydraulic excavators have increased since 2004, and market trends confirm a growing demand for the next years [112], which also means a rising demand for qualified excavator operators. In this scenario, simulation-based training combined with virtual reality is becoming a competitive alternative to traditional training to reduce costs and risks in the operator instruction.

Several excavator simulators for training purposes have been developed. For example, Caterpillar offers training simulators for several types of construction machines [17]. Its hydraulic excavator simulator features several training modules (bucket positioning, loading, trenching, ...) and provides feedback about elapsed time and accuracy in each training exercise; the simulator runs in a PC with a standard monitor, and includes two control levers as input devices. At the Institute of Robotics Research in Germany (IRF), Freund [38] developed an excavator simulator based on virtual reality technology with close-to-reality presentation of the environment and a physically based simulation; this simulator was aimed at commanding construction machines in real world applications by means of projective virtual reality and telepresence systems. Virtual reality technology was also used by Fukaya [40] to develop an immersive excavator simulator featuring 8 surrounding screens and an operator seat with a motion mechanism, in order to investigate the human factor of the operator and to find safety measures for excavator work. Torres [106], [107] developed a haptic interface-based simulator of a semiautomatic hydraulic excavator 2D arm in a virtual environment for training purposes; the goal was the kinesthetic coupling between the operator and the machine, providing information to the operator about the interaction forces between the excavator and the soil.

Research on excavator control and automation has also proposed kinematic and dynamic models for the mechanical behavior of hydraulic excavators. This research focused on designing control systems that can operate the excavator to perform certain tasks (e.g., generate straight-line motions in the bucket) with minimal human intervention, in order to increase productivity in the digging and ground leveling processes. The models developed in this field are also relevant and useful in a training simulator. Kinematic and physics-based dynamic models of the excavator arm, regarded as a planar manipulator with three degrees of freedom, were derived by Zhang [61], Hall [58] and Zweiri [114]. Makkonen [78] combined Matlab/Simulink and the commercial multibody simulation software *MSC.Adams* to develop a 3D model of the excavator arm with 4 degrees of freedom (DOF) equipped with a 2 DOF accessory at the end of the arm; while this model is significantly more complete than the previous ones, the simulations cannot run in real-time. Some authors included the hydraulic actuator circuit in the dynamic model of the excavator arm, see e.g. Chang [18] and He [60]; this approach takes into account the severe nonlinearities in hydraulic actuators, which are hardly observed in electric motors and make more difficult the design of control systems for automated excavators. Other complex phenomena present during excavator work have been also modeled: for example, Towarek [108] studied the interaction between an excavator with three-dimensional motion and a deformable soil foundation, and Coetzee [20] developed 2D discrete and continuum models of excavator bucket filling.

The aforementioned simulators for training purposes feature dynamic models of the excavator simple enough to run in real-time on desktop PCs, but they cannot simulate risky maneuvers like those shown in Figure 6.3. On the other hand, models originated from control research are limited to 2D arm models or they are too computationally expensive to run in a real-time training simulator. These limitations motivated the development of an excavator simulator with an improved dynamic model capable to accurately simulate all kind of maneuvers, accidents and dangerous situations, but fast enough to run in real-time in desktop computers. In addition, the simulator provides a low-cost virtual reality immersive environment to the operator, and therefore it is a powerful but affordable training tool for excavator operators.

A training simulator shall feature a dynamic model of the excavator that com-

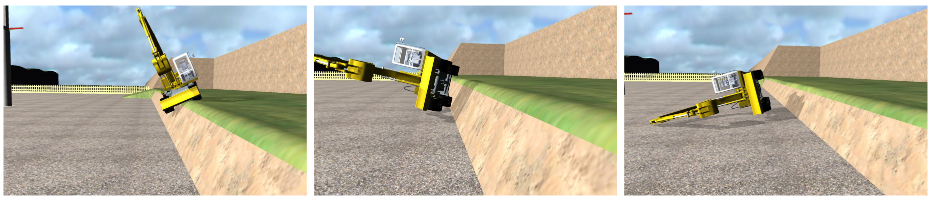


Figure 6.1: Rollover simulation

bines realistic physics-based behavior with high computational efficiency. The multibody approach described in this document fits this purpose, since the presented simulator can perform all kind of maneuvers that an operator can do in a real environment. The machine can sit on any combination of the blade, the outriggers, any or all the wheels, and the bucket. In addition of interacting with the fixed scenery, the machine can also collide with mobile targets as plastic fences delimiting operating zones.

6.1. Multibody model

The modeled machine is a Liebherr A924 Litronic, a medium-size wheeled excavator. It has been modeled with 14 rigid bodies and 13 revolute joints, shown in Figure 6.2. Elements crucial for stability like the front stabilizer blade and the left and right lateral outriggers (rear retractable legs) have been included in the model. Hydraulic cylinders have been modeled as kinematic constraints, since the dynamics of the hydraulic circuit has not been considered in this version of the simulator.

Kinematics of the multibody system has been modeled with natural coordinates [69]: this technique uses a set of dependent coordinates (points, vectors, angles and distances) related by constraints to characterize the motion of each body. The resulting excavator model has 154 coordinates (including 6 distances and 7 angles) and 154 constraints (10 of them are redundant).

The excavator model has 17 degrees of freedom (DOF), shown in Table 6.1 7 DOF are controlled by the operator, while the remaining 10 DOF are free. The actuated DOF for the hydraulic actuator circuit are kinematically guided; therefore, the operator controls the position of those actuated DOF without any delay or inertial effects. Velocities and accelerations of these kinematically guided DOFs have been adjusted to match the technical specifications of the real machine (torques and lift capacities). The motion of the non-actuated DOF is determined by the forces applied to the model:

- Weight of the machine parts and the bucket load.

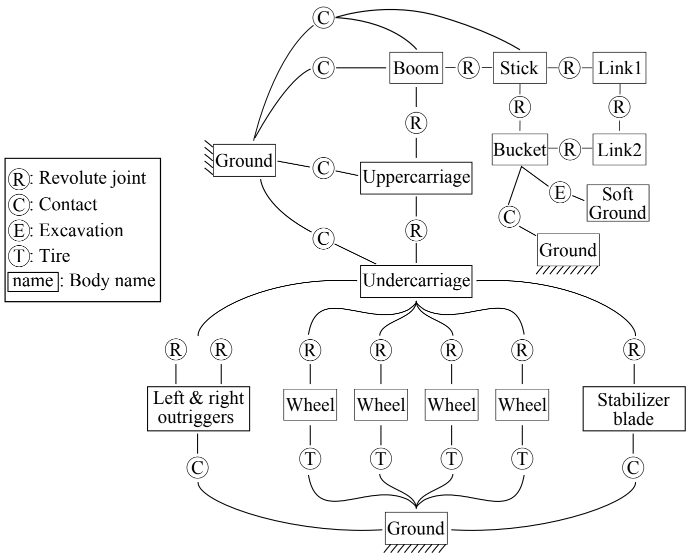


Figure 6.2: Multibody system topology for the excavator system

Motion	No.
Actuated degrees of freedom	
Boom, stick and bucket hydraulic cylinders	3
Uppercarriage rotation	1
Steering	1
Stabilizer blade	1
Outriggers	1
TOTAL	7
Non-actuated degrees of freedom	
Undercarriage free motion	6
Wheel rotation	4
TOTAL	10
TOTAL	17

Table 6.1: DOF listing

- Tire contact forces, which consist of linear spring and damper elements for the normal forces, and the magic formula tire model for the tangential forces [86].
- Tire torques applied with the accelerator and brake pedals.
- Contact forces originated from the collision of the excavator with the terrain or the surrounding objects.

6.2. Dynamic formulation

The equations of motion of the whole multibody system are given by the index-3 Augmented Lagrangian formulation described in §2.2.2; as integration scheme, the implicit single-step trapezoidal rule has been adopted, and cleaned velocities and accelerations are obtained by means of mass-damping-stiffness orthogonal projections (§2.3.1). The formulation is able to achieve real-time simulation of the excavator with time-steps of 5 ms. The multibody subroutines were written in the Fortran 2003 language.

6.3. Interaction with the environment

The excavator is placed in a working environment where the operator can perform different training exercises: maneuvering, digging, material handling, etc. The excavator interacts with the environment in two ways: (a) collisions with the scene objects and the terrain, which generate contact forces; and (b) terrain excavation and loading with the bucket. Some scene objects are fixed (e.g. buildings, terrain) while others are movable (e.g. fences). In order to compute the dynamics of movable objects, they are introduced in/removed from the simulation only when the excavator approaches to/moves away from them; this technique makes possible to simulate in real-time working environments with a large number of movable objects.

6.3.1. Collision detection

The collision detection system carries the duties of detecting and characterizing the possible contacts of the machine with the fixed scenery, so the forces coming from the contact of the tires, the outriggers, or the bucket against the terrain can be computed. Impacts or collision of the cabin and the scenery is also checked for logging purposes. An octree subdivision process as described in §4.4.1 speeds up the aforementioned tests. Several spheres are placed at different parts of the machine as the bucket or the wheels. At each time step, those spheres are tested against the octree, and if it exists a contact, the surface normal and indentation measure are passed to the corresponding force model.

For moving objects such as fences, the bounding sphere of the whole excavator, Figure 4.8 is tested against the ones for the fences. If a test is positive, the bounding spheres of the parts of the machine are tested against the AABB of the fence object so its reaction force can be calculated.

Finally, the bucket is tested to know if it lies within the boundaries of any deformable mesh object. Should this happen, the algorithm described in §4.8.1 is used to represent the digging operation.

6.3.2. Contact Model

Once a contact has been detected between the machine and the environment, contact forces shall be applied to the colliding bodies. The normal component of the contact forces is a regularized model with a combination of nonlinear spring and damper elements; this kind of models relates the energy dissipation to the impact velocity and the theoretical coefficient of restitution. Both the Lankarani-Nikravesh [75] and the Hunt-Crossley [67] models have been implemented in the simulator, obtaining similar results. The tangential-friction component of the contact force has a nonlinear Coulomb friction with sticking and sliding states. To model the frictional effects at low speed, characterized by the stiction state and the decreasing friction with increasing velocity, a model with elastic bristles was implemented; the smooth transition between stick and slip friction states was achieved by introducing the state function proposed by Gonthier [48]. The selected contact model delivers very realistic behavior and is able to simulate common events in the daily work of real excavators: slipping on slope terrains, stabilizing the machine with the blade or the outriggers, using the arm for support or impulsion (Figure 6.3), moving objects with the bucket, etc.

6.3.3. Terrain excavation and earth-moving operations

Excavation and earth loading are the most common tasks for excavators, and therefore they shall be included in the capabilities of a training simulator. The detailed simulation of bucket filling requires complex models to predict the material flow, see e.g. Coetzee [20]; this kind of models are too complex to run in real-time, and therefore the simplified bucket filling model described in §3.4 was developed.

The model includes digging forces that could even drag the machine if the

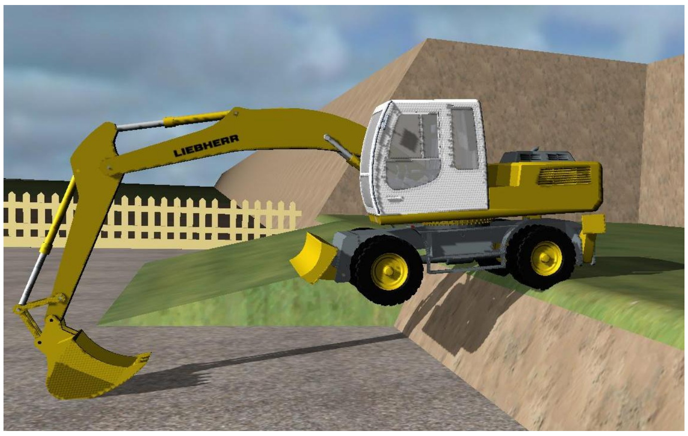


Figure 6.3: Using the arm for descending a steep slope

stabilizers are not active. That realistic event forces the simulator's users to remember to anchor the machine before digging operations.

6.4. Human-Machine interfaces

The operator console has a semi-immersive virtual reality interface that emulates the excavator cabin. A hard shell hemispherical dome of 2130 mm diameter from *Immersive Display UK Ltd.* is used to project the subjective view from the operator's position. It features an *Epson EMP-765C* projector and an *Omnifocus* lens that provides a $180°$ horizontal $\times$ $135°$ vertical view angle with XGA resolution (1024×768) at 72Hz. The OpenSceneGraph software library is used to render the virtual scene; the distortion correction for the hemispherical screen is achieved by the cube-mapping algorithm in §5.2.4.

The OpenAL library is used to generate spatial sound for the excavator engine, buzzers and collisions against objects in the scene.

6.4.1. Input controls

The operator console of the simulator emulates most of the controls in the real machine cabin using low-cost standard USB input devices: a steering wheel, 2 joysticks with the standard excavator functions (arm motion and uppercarriage rotation) and 2 pedals (accelerator and brake). In addition, a 15" LG L1510BF tactile screen (Figure 6.4) replicates the digital control panel of the excavator, which lets the operator control different machine settings (engine revolutions, drive speed, etc.) and shows warnings and errors. Some controls that exist as hardware switches in the actual excavator, like the ones to position the stabilizer blade

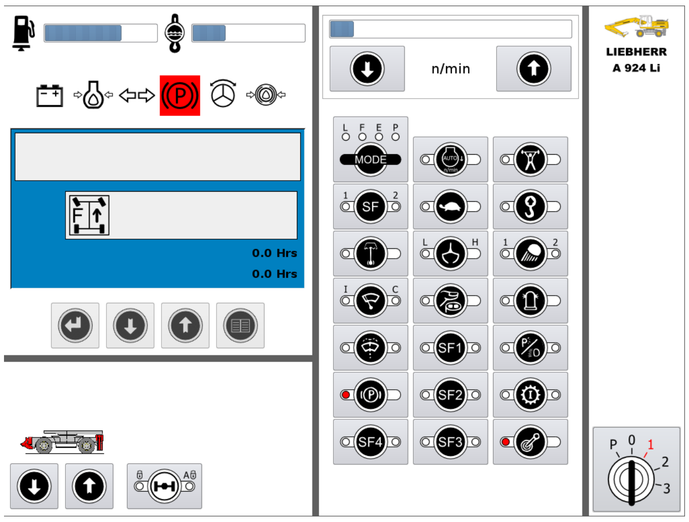


Figure 6.4: Tactile control panel

and outriggers, have been also included in the tactile screen as software switches, since they cannot be easily reproduced with standard off-the-shelf hardware.

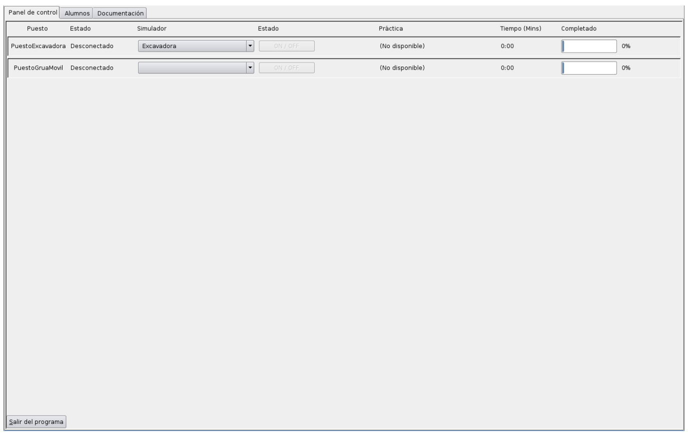
6.4.2. Monitoring

In addition to the operator console, the training simulator includes an instructor console: from this console, the instructor can control a networked group of operator consoles in a classroom to launch exercises, monitor the progress of the learners and evaluate them in a qualitative manner. The instructor console monitor features two cameras for the virtual simulator scene: a subjective view from the operator's point of view and a configurable external view. The instructor console also shows real-time information about events happened during the simulation (collisions, loss of stability, etc.).

The graphical interface of the program is subdivided into several parts. Each one is associated to a specific duty: a control panel (Figure 6.5), a student tracking module (Figure 6.6) and a documentation reader (Figure 6.7).

Control panel

The first tab in the interface displays a simulation seat control. From this tab, network-attached simulator seats can be displayed, queried for details, and controlled. A row is rendered for every simulator system on the local network. Each row displays several fields showing the status of that system and enabling some control widgets to perform session control:



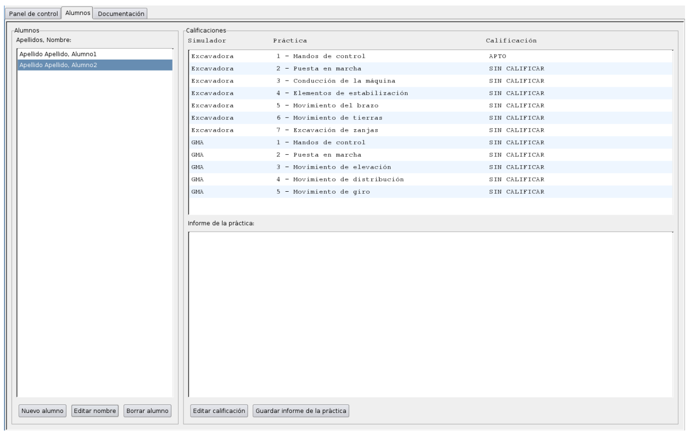
Puesto	Estado	Simulador	Estado	Práctica	Tiempo (Mins)	Completado
PuestoExcavadora	Desconectado	Excavadora	ON / OFF	(No disponible)	0:00	0%
PuestoGruaMovil	Desconectado		ON / OFF	(No disponible)	0:00	0%

Figure 6.5: Seat monitoring tab

- Simulator system name: a descriptive name which helps to identify the real simulator system.
- State field: there, it is shown whether the simulator was detected on the local network by means of direct connection. When the simulator starts, it opens a listening port on the network interface which can be used later for communication purposes with the instructor console.
- Simulator name: a drop-down list where the available machine simulations are shown. The instructor console is not tied to a particular simulator, and can be used to monitor several different kinds of machine simulators, not only the excavator simulator discussed in this chapter.
- State button: this button can be used to start or finish a session by clicking on it.
- Session number: a drop-box list where the monitor can choose the desired session to be simulated.
- Time: a field displaying the elapsed time since the session was started.
- Completion status: a graphical bar showing the progress being made as a percentage meter. The progress information is retrieved from the script of the training session.

Student tracking module

The tab for this module displays the contents of a database containing simulator systems' student information. On a first list, names of the students are



Apellidos, Nombre:

Apellido Apellido, Alumno1
Apellido Apellido, Alumno2

Simulador	Práctica	Calificación
Excavadora	1 – Mandos de control	APTO
Excavadora	2 – Puesta en marcha	SIN CALIFICAR
Excavadora	3 – Conducción de la máquina	SIN CALIFICAR
Excavadora	4 – Elementos de estabilización	SIN CALIFICAR
Excavadora	5 – Movimiento del brazo	SIN CALIFICAR
Excavadora	6 – Movimiento de tierras	SIN CALIFICAR
Excavadora	7 – Excavación de zanjas	SIN CALIFICAR
GMA	1 – Mandos de control	SIN CALIFICAR
GMA	2 – Puesta en marcha	SIN CALIFICAR
GMA	3 – Movimiento de elevación	SIN CALIFICAR
GMA	4 – Movimiento de distribución	SIN CALIFICAR
GMA	5 – Movimiento de giro	SIN CALIFICAR

Informe de la práctica:

Figure 6.6: Student tracking tab.

written on the screen. Those names can be clicked, and a list of the available sessions are presented on the right of the window. The session listing shows the simulator type they belong to, their names and the current qualification of the student, should it exists. In a edit control below, the instructor can take some notes about the performance of the simulation session if necessary.

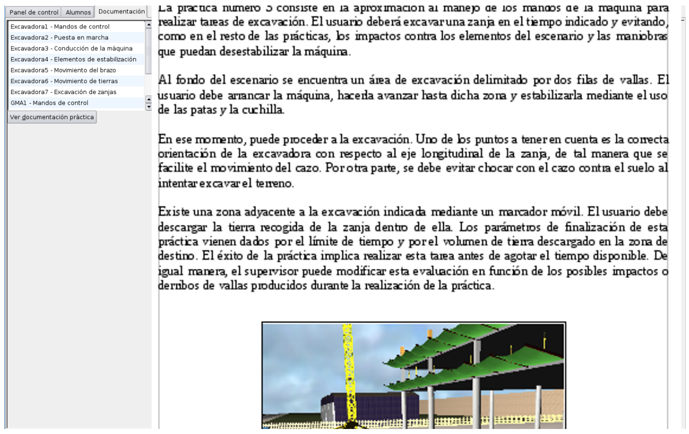
Documentation reader

The last tab in the instructor console contains the reference manuals for the available simulation sessions. On the left part of the screen, a list of the sessions are listed. On the right, a document area shows the documentation describing the session. The documentation can be presented in PDF format, which will be later converted to the graphical output that it is presented to the instructor. The documentation presents the task to be carried, the objectives to be performed, and the possible constraints or mistakes to avoid.

Run-time monitoring tracker

As soon as the instructor selects a session and commands a simulator to run it by means of pressing the *ON* button, a new tab is created in the console in order to be able to track the progress of the student during task realization.

Support for different machine simulators is implemented by a plugin system, each one of them sharing a common interface. The interface allows the instructor console to communicate with each type of simulator, and correctly interpret and display the data they are sending. For example, all the plugins implement a command to render the simulation scene, but it is up to each plugin how to draw


Panel de control
Alumnos
Documentación
Excavadora1 - Mandos de control
Excavadora2 - Puesta en marcha
Excavadora3 - Conducción de la máquina
Excavadora4 - Elementos de estabilización
Excavadora5 - Movimiento del brazo
Excavadora6 - Movimiento de tierras
Excavadora7 - Excavación de zanjas
GMA1 - Mandos de control
Ver documentación práctica

La práctica número 3 consiste en la aproximación al manejo de los mandos de la máquina para realizar tareas de excavación. El usuario deberá excavar una zanja en el tiempo indicado y evitando, como en el resto de las prácticas, los impactos contra los elementos del escenario y las maniobras que puedan desestabilizar la máquina.

Al fondo del escenario se encuentra un área de excavación delimitado por dos filas de vallas. El usuario debe arrancar la máquina, hacerla avanzar hasta dicha zona y estabilizarla mediante el uso de las patas y la cuchilla.

En ese momento, puede proceder a la excavación. Uno de los puntos a tener en cuenta es la correcta orientación de la excavadora con respecto al eje longitudinal de la zanja, de tal manera que se facilite el movimiento del cazo. Por otra parte, se debe evitar chocar con el cazo contra el suelo al intentar excavar el terreno.

Existe una zona adyacente a la excavación indicada mediante un marcador móvil. El usuario debe descargar la tierra recogida de la zanja dentro de ella. Los parámetros de finalización de esta práctica vienen dados por el límite de tiempo y por el volumen de tierra descargado en la zona de destino. El éxito de la práctica implica realizar esta tarea antes de agotar el tiempo disponible. De igual manera, el supervisor puede modificar esta evaluación en función de los posibles impactos o derribos de vallas producidos durante la realización de la práctica.

Figure 6.7: Training session documentation tab.

the special features of its scenery, as the deformable terrain, or the cables of the machinery, or other needed special effects.

The interface for this tab consists on two 3D views depicting the same scenario of the simulation session. One view presents the point of view of the person performing the simulation, and the other lets the instructor to navigate at will through the operation space, making it possible to view the maneuvering from any point.

There are also available a timer and a progress meter displaying the course of the session.

In the lower part of the panel, there is a text field displaying the log for the session, with every event starting by a time stamp, and a textual description of it. Events can be normal simulation events as the starting and ending of a session, or messages from the simulator warning about errors or mistakes committed by the user.

6.5. Tasks & duties definition

A key point about simulators, once a realistic behavior and scenery has been attained, is the possibility of being able to define training sessions in a straightforward manner. Usually the task of designing the sessions is not carried over by the same team that developed the simulator. Experienced personnel in the training field have a better view on the added value of any session that can be defined for the simulator. Therefore, it is very helpful to have a high-level, intuitive system that can be used mainly by non-programmers and that does not require to rebuild the code in order to add new sessions to the simulator.

Scripting languages have been used traditionally for extension purposes on software products. Their simplicity makes them ideal for people with lesser or minimal programming skills, presenting a quite low learning curve. Usually those languages do not require special tools apart from a text editor, if it is not already integrated into the software. Scripting makes possible to define tasks and algorithms that cannot be expressed only by means of parameter specification into configuration files, and therefore can adapt their behavior at run-time. Scripted code can be used to define any kind of behavior or algorithm that any user could need, relieving the simulator developers from the job of foreseeing them, should it be possible. General purpose languages like most scripting languages are *Turing complete* or *computationally universal*, meaning that a set of rules can always be found in order to perform any calculation: the scripting environment provides the necessary tools so the users can write code that implements any kind of behavior not devised by the simulator developers.

As an example, a script could connect to an external database in order to read updated data for the messages exposed to the user, or any other parameter employed into the simulation.

However, the integration of scripting libraries into the software needs to design a fixed communication interface between them. The simulator code defines what information must expose to the script and what data should expect to be returned when the script call finishes. This is the only hard requirement of the scripting system, because the specific information available to the script from the simulator has to be stated explicitly when it is copied between both environments.

Another advantage of scripting languages is that they benefit from a vast collection of software modules written in order to carry a huge number of common tasks, lessening the chances of having to write them from scratch. Following the example of the script contacting to an external database, that communication can be done with the aid of existing networking and database accessing modules.

6.5.1. Python scripting language integration

For the excavator simulator, the Python scripting language was chosen. Among its advantages, it is a widespread and mature language, and it is available in virtually all computing platform systems. Its free license makes it attractive for a large user community, and therefore it is easy to find new modules and libraries for developing any kind of computational task, like database manipulation, network communication, graphical user interface creation, file and system handling, numerical computation, etc.

Another big advantage of the Python language is its the easiness of embedding script code into existing C or C++ programs. There are simple functions that allow a program to create, modify or query for different type of variables in the script. Therefore, the whole data that is managed by the script is available at any time from the main program. This fact eases communication procedures for passing data between the program and the script.

A common characteristic of scripting languages is to have *introspection* characteristics. Introspection is the capability that allows the user to query any object or symbol present in the scripting environment in order to know its type and other useful details. In fact, the scripting language environment can be queried for the current existing functions and variables, report their values or modify them. The main program can check at run-time that all the necessary variables and functions required by the simulator are defined, avoiding the recompilation of the software, and postponing the design stage for the simulator tasks, as described earlier.

Python embedding

The Python language provides development modules that are used in the simulator to query and interact with user scripts. Those modules are C headers and libraries that implement functions that allow to read and modify script data, and run script functions. Most of the interconnection functions are defined in terms of a generic type, `PyObject`. This type represents any type of object — data variable, module or function — that could exist in the scripting environment. By using the introspection mechanism, the simulator can query its actual type, name and value. Therefore, the simulator can look for specific variables and functions in order to run the task control.

The simulator features specific code to load a Python script file, and execute some of the functions written inside. Initially, the simulator loads the script file that it is commanded to run for the simulation task to be performed. The name of the script file is given by the start-up command for the simulator, which has been issued from the control panel of the monitor described in reference.

```
pModule = PyImport_Import(pName);
```

Function `PyImport_Import` loads a file with a name specified by `pName`, and returns a `PyObject` handle, needed for further operations. The handle represents a module that encloses all the data and the code present in the script file.

For convenience, code loading and running a certain function in the module is encapsulated into a C++ class in order to reuse common code.

The subroutine `PyObject_GetAttrString` can be used to retrieve the function from the module by its name. Its input arguments are the module handle, and the name of the scripting function. It returns a `PyObject` variable pointing to the function object. It can be additionally assured that the handle corresponds to a callable object using the function `PyCallable_Check`.

For each different training session, the provided script has to implement at least two well-known functions: an initialization function and a practice training function. Those functions define the aforementioned interface between both simulator components.

The initialization function of the script is required to serve as the point of execution in the program where all the global parameters for the simulator are set. Those parameters are either needed for starting the simulation or either are not going to change during the whole session.

Common initial starting parameters are the name of the three-dimensional geometry scenery file, the position of the mobile objects over it — including the machine itself — and other visual cues, as marks over the terrain or the summary text of the exercise to be initially read by the simulator's users.

Script interface implementation

The implementation is done in the following way: user's initialization function has to create a Python *dictionary* holding different pieces of information to pass to the simulator. A *dictionary* is a special type of variable in Python that implements the concept of an *associative container.* An *associative container* is a variable that holds several pairs of pieces of data. For each pair, the first piece of data is called the *key*, whereas the second is referred as the *value.* Data values can be read from the dictionary by querying by its associated *key.* The data type of the *key* can be almost anything, but a common technique is to use text strings, in order to store data values by their name. The *values* can also adopt any data type, such as numbers or strings, or even lists and dictionaries. Therefore, it is very convenient to locate any desired parameter in the script just by its name specification.

```
In [1]: dictionary={"Excavator_Position":(0,0,0), "Available_Time":120}

In [2]: dictionary["Available_Time"]
Out[2]: 120
```

The dictionary returned by the initialization script function can store the following specific parameters:

- Excavator position: a list of three floating points specifying the initial position of the machine in the scenery.
- Fence information: the simulator features some movable objects, fences, that can be placed into the scenery in order to mark a path or as obstacles to be avoided. At a final stage, their positions can be checked in order to

know if the machine collided with them, and compute the final scoring accordingly. This object stores a list of positions and orientations for the fences, if available.

- Deformable objects information: diggable objects can be specified and placed into the simulation by means of this parameter. They are encoded as lists of their (x, y) coordinates over the ground.

- Briefing: a text string holding the description, objectives and tips for the task to be simulated. It will be presented to the user at the beginning of the session.

- Target mark positions: when needed, some visible marks can be placed on the scenery in order to signal where a certain event is going to take place. For example, the user can be required to move the machine to the proposed mark, or dump the excavation material there. They are encoded as lists of their (x, y) coordinates over the ground.

- Bucket load: the initial amount of material carried into the machine's bucket. It is considered empty if this parameter is not specified, otherwise the floating point value is used to set its initial load.

As discussed earlier, the introspection mechanism makes it possible to check at run-time if a certain variable or parameter is defined, therefore allowing the use of optional parameters in the established interface. The bucket load, for example, can be safely ignored by the script writer if it is not desired to have any, simplifying the initialization process by avoiding to define safe default values for unused parameters.

The in-practice-time scripting function receives some real-time data information about the current state of the simulator. Nevertheless, it is not necessary to execute the script at each integration step, and therefore it is called just before the rendering step seen in Figures 5.2 and 5.3 from §5.2.1. The script function must return information about the course of the practice. Input values for the script are:

1. 3D position of the excavator on the ground.

2. 3D position and orientation of the mobile objects in the scene, the plastic fences. The original position of every object is also provided, so an estimation of if any of them has been moved can be performed.

3. Position of each diggable surface and the volume it comprises.

4. Elapsed simulation time.

After that, the script must evaluate those parameters and return a list containing the completion percentage of the task, the error or success status, and an error message —should it be necessary to warn the user about an unsuccessful session completion. The simulator can display and transmit the state of the session, know if the session is over and what message to display to the user.

As mentioned, the completion percentage of the task is computed according the current script function for the session. Therefore it is completely user-defined, since the criterion is defined by the code of that function. The script can compute the updated session progress based on the current machine position, the amount of material dug from a heap of material, or dumped to a certain location: it is absolutely customized by the script's writer. The simulator also sends progress information to the monitoring seat, in order to display there the current status of the session.

The status parameter shows if there exist any event to signal during the session. There exist three states: no error, warning, and session end. The warning state lets the script notify the simulator that a special event has been triggered and that it is worth mentioning it either to the user, the monitor or both. This is typically used to warn about collisions against elements of the scenery or other non-fatal mistakes that the driver could commit. A warning text is also sent to the monitor in order to fill the session's log with relevant information about the course of the training. Finally, the session end state indicates that the simulation is over, and if it was successful or not. This state makes the simulator to stop the session and to notify it to the monitoring seat.

In order to write the scripts, some utility modules are provided to aid the session designers to place easily the objects over the terrain, and to position and orient the fences. Some basic algebra functions are also provided, even though they could be also included into the script by importing any of the available Python packages. The effect of having those features is to help less experienced users in case they had no extra modules available in their systems. This eliminates the need of having to install extra software, if possible. Nevertheless, as noted previously, a more experienced designer can take full advantage of out-of-the-shelf software, to write more complex scripts. Again, the scripting environment allows to extend or customize the program in ways that could not have been devised when it was originally developed.

Chapter 7

Conclusions

This thesis aims to establish a convenient framework for developing multibody-based machinery simulators. Recurrent problems that appear when developing a simulator of this kind were addressed.

- In Chapter 2, a sensible combination of multibody formulation, coordinates and integration method were described. This system is fast enough to be used in interactive simulators, while at the same time it is flexible enough for being able to enlarge or shrink the number of simulated bodies without high penalties.
- Chapter 3 described a series of contact models, including the computation of reaction forces and friction forces, that are suitable to be used at interactive rates, yet preserving a realistic accuracy. A deforming terrain model including drag forces was presented in order to simulate excavation operations.
- Chapter 4 discussed a set of algorithms 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 shown, and several detailed collision detection models were suggested depending on the specific characteristics of the geometry of the bodies in contact. The special case of a deforming mesh for simulating terrain manipulation was also described.
- In Chapter 5 hardware devices for displaying the output of a simulator, and to emulate the real controls of a machine were shown, emphasizing the use of COTS parts in order to lower the costs of the simulator's development. Several software guidelines were presented in order to demonstrate the use of curved-surface screens, or networking devices for communication purposes.

- Chapter 6 shows the detailed description of an excavator simulator, including the implementation of a scripting system for describing and creating new training sessions, and the remote monitoring and grading system for managing and evaluating each one of them.

Real-time simulation techniques from multibody system dynamics allowed to develop a realistic but computationally efficient physics-based model of a complex machine like a hydraulic excavator. The motion equations where expressed in an index-3 Augmented Lagrangian formulation and integrated with a fixed time step of 5 milliseconds. A Hunt-Crossley model for computing reactions coming from contact events was used, coupled with a tangential force model for modeling friction and stiction phenomena. A terrain model running at interactive rates was developed for the simulation of earthmoving operations of the machine. Spatial partitioning techniques were implemented for a fast determination of the colliding objects in the scene. Mobile objects in the scene are added or removed from the multibody system at run-time in order to save computational resources and being able to simulate a large number of objects. Complex three-dimensional maneuvers as rollovers, or impulsion with the bucket or the support legs can be performed, due to the generic multibody model.

The simulator system was equipped with industrial-quality controls and tactile panels that mimic the real control panel of the machine. An immersive environment was built by means of a graphics projection over a dome-shaped screen and real audio output. The program can run on a non-expensive standard PC computer: it was tested on an average domestic computer (Intel Core i7 920 processor at 2.67 Ghz) and even on some laptops.

The framework presented on this thesis constitutes a good starting point for the development of new machinery simulators. However, there are still some additional research lines that could help to improve their overall quality.

7.1. Future lines of research

7.1.1. Simulation Parallelization

Multibody methods presented so far do not impose any particular software implementation guidelines. The simulators 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 shown in §§5.4 and 6.3.1 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. [49] warns about the high penalty that small or medium-sized multibody problems incur into when trying to parallelize the multibody systems: usually the bottlenecks are computation of the jacobian of the constraints and the solving of the final linear equation system. However, their size is too small for not having significant synchronization costs.

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 Communications* for data passing between the processes.

Parallelization also eases co-simulation tasks with other non-multibody simulator systems (hydraulic, electronic...), due the loose coupling that this strategy imposes.

7.1.2. Granular Media Simulation and Interaction

In this thesis a model for representing the interaction of a machine and a granular media composed soil was described. Forces resulting from the movement of the machine's bucket inside a heap of granular media were computed based on the depth and the flow of material going inside of the bucket.

Current research on the motion of granular media is done following the *Discrete Element Method* [26]. The model uses simple equations for each one of the grains, and therefore is well behaved for its computation in highly-parallel devices such as GPUs or clusters of computers. However, the costly (in time) data transfers between the processing unit doing the granular media computation and the main unit running the simulation can not be fast enough for sustaining interactive rates.

Using simplified granular models that run at a higher time step than the main simulation could compensate the time spent during data transfers back and forth to the computing device, at the cost of a diminished accuracy in the terrain model. For example, if the main simulation integrator runs at a time step of 1ms, the terrain model could run at 10ms on the GPU and devote the rest of the time to synchronize the data with the main loop every 10 time steps. Therefore, the current digging volume computation can be improved by using a more accurate algorithm.

Bibliography

[1] U.M. Ascher and L.R. Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations.* Philadelphia Society for Industrial and Applied Mathematics, 1998.

[2] R. Azuma et al. “Recent Advances in Augmented Reality”. In: *IEEE Computer Graphics and Applications* 21.6 (2001), pp. 34–47.

[3] Bruce G. Baumgart. “A polyhedron representation for computer vision”. In: *Proceedings of the May 19-22, 1975, national computer conference and exposition.* AFIPS ’75. Anaheim, California: ACM, 1975, pp. 589–596.

[4] J. Baumgarte. “Stabilization of constraints and integrals of motion in dynamical systems”. In: *Computer Methods in Applied Mechanics and Engineering* 1 (1982), pp. 1–16.

[5] E Bayo and A Avello. “Singularity-Free Augmented Lagrangian Algorithms for Constrained Multibody Dynamics”. In: *Nonlinear Dynamics* 5.2 (Mar. 1994). Times Cited: 13 Article English BAYO, E UNIV CALIF SANTA BARBARA,DEPT MECH ENGN,SANTA BARBARA,CA 93106 Cited References Count: 37 NP283 KLUWER ACADEMIC PUBL SPUIBOULEVARD 50, PO BOX 17, 3300 AA DORDRECHT, NETHERLANDS DORDRECHT, pp. 209–231.

[6] E. Bayo, J. García de Jalón, 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. 1988), pp. 183–195.

[7] 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.

[8] J. L. Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517.

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

[10] K.D. Bhalerao, K.S. Anderson, and J.C. Trinkle. “A Recursive Hybrid Time-Stepping Scheme for Intermittent Contact in Multi-Rigid-Body Dynamics”. In: *Journal of Computational and Nonlinear Dynamics* 4.4 (Oct. 2009).

[11] Boeing. *Boeing Simulator Services.* 2013. URL: `http://www.boeing.com/commercial/aviationservices/flight-services/simulator-services/index.html`.

[12] Boost.org. *Boost Serialization Library.* URL: `http://www.boost.org/libs/serialization`.

[13] P. Bourke. “Using a spherical mirror for projection into immersive environments”. In: *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia.* Graphite (ACM Siggraph). Dunedin, 2005, pp. 281–284.

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

[15] D. C. Brooker. “Numerical Modelling of Pipeline Puncture under Excavator Loading, part I. Development and Validation of a Finite Element Failure Model for Puncture Simulation”. In: *International Journal of Pressure Vessels and Piping* 80.10 (2003), pp. 715–725.

[16] 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, Transactions ASME* 67.4 (2000), pp. 831–834.

[17] Caterpillar. *CAT Virtual Training System Simulators.* Caterpillar. http://www.cat.com.

[18] P. H. Chang and S. J. Lee. “A Straight-Line Motion Tracking Control of Hydraulic Excavator System”. In: *Mechatronics* 12.1 (2002), pp. 119–138.

[19] 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.

[20] C. J. Coetzee, A. H. Basson, and P. A. Vermeer. “Discrete and Continuum Modelling of Excavator Bucket Filling”. In: *Journal of Terramechanics* 44.2 (2007), pp. 177–186.

[21] Richard W. Cottle and George B. Dantzig. “Complementary pivot theory of mathematical programming”. In: *Linear Algebra and its Applications* 1.1 (1968), pp. 103–125.

[22] Carolina Cruz-Neira et al. "CAVE. Audio visual experience automatic virtual environment". In: *Communications of the ACM* 35.6 (1992), pp. 65–72.

[23] J. Cuadrado et al. "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.

[24] J. Cuadrado et al. "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.

[25] Javier Cuadrado et al. "Automotive observers based on multibody models and the extended Kalman filter". In: *Multibody System Dynamics* 27 (1 2012), pp. 3–19.

[26] P.A. Cundall and O.D.L. Strack. "Discrete Numerical Model for Granular Assemblies". In: *Geotechnique* 29.1 (1979), pp. 47–65.

[27] Adam Czaplicki, Miguel T. Silva, and Jorge C. Ambrósio. "Biomechanical Modelling for Whole Body Motion Using Natural Coordinates". In: *Journal of Theoretical and Applied Mechanics* 20.4 (2004), pp. 927–944.

[28] S. Djerassi. "Collision with friction; Part A: Newton's hypothesis". In: *MULTIBODY SYSTEM DYNAMICS* 21.1 (Feb. 2009), pp. 37–54.

[29] S. Djerassi. "Collision with friction; Part B: Poisson's and Stronge's hypotheses". In: *MULTIBODY SYSTEM DYNAMICS* 21.1 (Feb. 2009), pp. 55–70.

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

[31] U. Drepper. *What Every Programmer Should Know About Memory.* URL. Nov. 2007.

[32] E. Eich-Soellner and C. Führer. *Numerical Methods in Multibody Dynamics.* B.G.Teubner Stuttgart, 1998.

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

[34] 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. 2010), pp. 165–190.

[35] P. Flores et al. "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.

[36] P. Flores et al. *Kinematics and Dynamics of Multibody Systems with Imperfect Joints.* Springer-Verlag, 2008.

[37] J. D. Foley et al. *Introduction to Computer Graphics.* Addision–Wesley Proffesional, 1993.

[38] E. Freund, J. Rossman, and T. Hilker. "Virtual Reality Technologies for the Realistic Simulation of Excavators and Construction Machines: From VR-Training Simulators to Telepresence Systems". In: *Mobile Robots XV and Telemanipulator and Telepresence Technologies VII.* 2000.

[39] H. Fuchs, Z. M. Kedem, and B. F. Naylor. "On visible surface generation by a priori tree structures". In: *SIGGRAPH Computer Graphics* 14.3 (July 1980), pp. 124–133.

[40] K. Fukaya and S. Umezaki. "Development of Excavator Simulator using Virtual Reality Technology". In: *Virtual Reality Society of Japan Annual Conference.* 2002.

[41] J. C. García Orden and Daniel Dopico Dopico. "On the Stabilizing Properties of Energy-Momentum Integrators and Coordinate Projections for Constrained Mechanical Systems". In: *Multibody Dynamics.* Ed. by Juan Carlos García Orden, José M. Goicolea, and Javier Cuadrado. Vol. 4. Computational Methods in Applied Sciences. Springer Netherlands, 2007, pp. 49–67.

[42] W.Riley Garrott et al. "Methodology for validating the National Advanced Driving Simulator's Vehicle Dynamics (NADSdyna)". In: *SAE Special Publications* 1228 (1997), pp. 71–83.

[43] E.G. Gilbert, D.W. Johnson, and S.S. Keerthi. "A fast procedure for computing the distance between complex objects in three-dimensional space". In: *IEEE Journal of Robotics and Automation* 4.2 (Apr. 1988), pp. 193 –203.

[44] C. Glocker and F. Pfeiffer. "Multiple Impacts with Friction in Rigid Multibody Systems". In: *Nonlinear Dynamics* 7.4 (June 1995), pp. 471–497.

[45] David Goldberg. "What Every Computer Scientist Should Know About Floating Point Arithmetic". In: *ACM Computing Surveys* 23.1 (1991), pp. 5–48.

[46] W. Goldsmith. *Impact, The theory and physical behaviour of colliding solids.* London: Edward Arnold Ltd., 1960.

[47] Y. Gonthier et al. “A contact modeling method based on volumetric properties”. English. In: *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.* Vol. 6, Pts A-C. 5th International Conference on Multibody Systems, Nonlinear Dynamics, and Control, Long Beach, CA, SEP 24-28, 2005. ASME Design Engn Div; ASME Comp & Informat Engn Div. New York: ASME, 2005, pp. 477–486.

[48] Y. Gonthier et al. “A regularized contact model with asymmetric damping and dwell-time dependent friction”. In: *Multibody System Dynamics* 11.3 (2004), pp. 209–233.

[49] Francisco González et al. “Non–intrusive parallelization of multibody system dynamic simulations”. In: *Computational Mechanics* 44.4 (2009), pp. 493–504.

[50] F. González et al. “On the Effect of Multi-rate Co-simulation Techniques in the Efficiency and Accuracy of Multibody System Dynamics”. In: *Multibody System Dynamics* 25.4 (2011), pp. 461–483.

[51] Google. *Protocol Buffers homepage.* URL: `http://developers.google.com/protocol-buffers`.

[52] S. Gottschalk. *Separating Axis Theorem.* Tech. rep. TR96-024. Department of Computer Science, UNC Chapel Hill, 1996.

[53] S. Gottschalk, M.C. Lin, and D. Manocha. “OBBTree: A hierarchical structure for rapid interference detection”. In: 1996, pp. 171–180.

[54] Ned Greene. “Environment Mapping and other Applications of World Projection”. In: *IEEE Computer Graphics and Applications* 6.11 (1986), pp. 21–29.

[55] P. Guigue and O. Devillers. “Fast and Robust Triangle-Triangle Overlap Test Using Orientation Predicates”. In: *Journal of Graphics Tools* 8.1 (2003), pp. 25–32.

[56] M. Géradin and A. Cardona. *Flexible Multibody Dynamics. A Finite Element Approach.* Chinchester (England): John Wiley & Sons Ltd, 2001.

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

[58] A. S. Hall and P. R. McAree. “A Study of the Interaction Between Operator Style and Machine Capability for a Hydraulic Mining Excavator”. In: *Journal of Mechanical Engineering Science* 219.5 (2005), pp. 477–489.

[59] E.J. Haug. *Computer Aided Kinematics and Dynamics of Mechanical Systems.* Allyn and Bacon series in engineering vol. 1. Allyn & Bacon, Incorporated, 1989.

[60] Q. H. He et al. "Study on Motion Simulation of Hydraulic Excavator's Manipulator". In: *Xitong Fangzhen Xuebao / Journal of System Simulation* 18.3 (2006).

[61] Q.H. He et al. "Modeling and control of hydraulic excavator's arm". English. In: *Journal of Central South University of Technology* 13 (4 2006), pp. 422–427.

[62] Martin Held. "ERIT - A Collection of Efficient and Reliable Intersection Tests". In: *Journal of Graphics Tools* 2 (1998), pp. 25–44.

[63] M. Hereld, I.R. Judson, and R.L. Stevens. "Introduction to building projection-based tiled display systems". In: *Computer Graphics and Applications, IEEE* 20.4 (2000), pp. 22–28.

[64] 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.

[65] P. M. Hubbard. "Approximating Polyhedra with Spheres for Time-Critical Collision Detection". In: *ACM Transactions on Graphics* 15 (1996), pp. 179–210.

[66] S. Hummel, M. Kennedy, and E. Ashley Steel. "Assessing forest vegetation and fire simulation model performance after the Cold Springs wildfire, Washington USA". In: *Forest Ecology and Management* 287 (2013), pp. 40–52.

[67] K.H. Hunt and F.R.E. Crossley. "Coefficient of Restitution Interpreted as Damping in Vibroimpact". In: *Journal of Applied Mechanics, Transactions ASME* 42 Ser E.2 (1975), pp. 440–445.

[68] K.A. Ismail and W.J. Stronge. "Impact of viscoplastic bodies: Dissipation and restitution". In: *Journal of Applied Mechanics, Transactions ASME* 75.6 (2008), pp. 0610111–0610115.

[69] J. García de Jalón and E. Bayo. *Kinematic and dynamic simulation of multibody systems: The real-time challenge.* New York (USA): Springer-Verlag, 1994.

[70] S. Karmakar and R.L. Kushwaha. "Dynamic modeling of soil-tool interaction: An overview from a fluid flow perspective". In: *JOURNAL OF TERRAMECHANICS* 43.4 (Oct. 2006), pp. 411–425.

[71] Donald E. Knuth. *The art of computer programming: seminumerical algorithms.* 3rd ed. Vol. 2. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[72] Creative Labs. *OpenAL official website.* URL: `http://connect.creativelabs.com/openal`.

[73] Vortex CM Labs. *Vortex Simulators.* URL: `http://www.vxsim.com/en/simulators/index.php`.

[74] H.M. Lankarani and P.E. Nikravesh. "Contact force model with hysteresis damping for impact analysis of multibody systems". In: *Journal of Mechanical Design* 112.3 (1990), pp. 369–376.

[75] H.M. Lankarani and P.E. Nikravesh. "Continuous contact force models for impact analysis in multibody systems". In: *Nonlinear Dynamics* 5.2 (1994), pp. 193–207.

[76] E. Leylek, M. Ward, and M. Costello. "Flight dynamic simulation for multibody aircraft configurations". In: *Journal of Guidance, Control, and Dynamics* 35.6 (2012), pp. 1828–1842.

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

[78] T. Makkonen, K. Nevala, and R. Heikkilä. "A 3D Model Based Control of an Excavator". In: *Automation in Construction* 15.5 (2006), pp. 571–577.

[79] M. Mäntylä. *An Introduction to Solid Modeling.* W.H. Freeman & Company, 1988.

[80] Tomas Möller. "A Fast Triangle-Triangle Intersection Test". In: *Journal of Graphics Tools* 2.2 (1997), pp. 25–30.

[81] M. A. Naya et al. "An Efficient Unified Method for the Combined Simulation of Multibody and Hydraulic Dynamics: Comparison with Simplified and Co-Integration Approaches". In: *Archive of Mechanical Engineering* LVIII.2 (July 2011), pp. 223–243.

[82] Bruce Naylor, John Amanatides, and William Thibault. "Merging BSP trees yields polyhedral set operations". In: *SIGGRAPH Computer Graphics* 24.4 (Sept. 1990), pp. 115–124.

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

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

[85] N. Noury and T. Hadidi. “Computer simulation of the activity of the elderly person living independently in a Health Smart Home”. In: *Computer Methods and Programs in Biomedicine* 108.3 (2012), pp. 1216–1228.

[86] H. B. Pacejka and E. Bakker. “Tyre models for vehicle dynamics analysis”. In: H.B.Pacejka (ed.), Taylor and Francis, 1993. Chap. The Magic Formula Tyre Model.

[87] L.R. Petzold. “A Description of DASSL: A differential/algebraic system solver”. In: North-Holland, 1983, pp. 65–68.

[88] F. Pfeiffer. “Complementarity problems of stick-slip vibrations”. In: *Journal of Vibration and Acoustics-Transactions of the ASME* 118.2 (Apr. 1996), pp. 177–183.

[89] CH products. *Industrial-quality controls.* URL: `http://www.chproducts.com`.

[90] E. Rabinowicz. “Stick and slip”. In: *Scientific American* 194.5 (1956), pp. 109–118.

[91] Chris Robinson. *OpenAL software implementation website.* URL: `http://kcat.strangesoft.net/openal.html`.

[92] J. M. Rolfe and K. J. Staples, eds. *Flight Simulation.* Cambridge Aerospace Series. Cambridge University Press, 1988.

[93] A. Rouvinen, T. Lehtinen, and P. Korkealaakso. “Container Gantry Crane Simulator for Operator Training”. In: *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 219.4 (2005), pp. 325–336.

[94] W. Schiehlen and R. Seifried. “Three approaches for elastodynamic contact in multibody systems”. In: *MULTIBODY SYSTEM DYNAMICS* 12.1 (Aug. 2004), pp. 1–16.

[95] J. Schöberl. “NETGEN — An advancing front 2D/3D-mesh generator based on abstract rules”. In: *Computing and Visualization in Science* 1 (1997), pp. 41–52.

[96] MA Serna, R Aviles, and J García de Jalón. “Dynamic Analysis of Plane Mechanisms with Lower Pairs in Basic Coordinates”. In: *Mechanism and Machine Theory* 17.6 (1982). Times Cited: 7 Article English SERNA, M. A ESCUELA SUPER INGN IND,CATEDRA MECAN,BILBAO,SPAIN Cited References Count: 21 PV024 PERGAMON-ELSEVIER SCIENCE LTD THE BOULEVARD, LANGFORD LANE, KIDLINGTON, OXFORD, ENGLAND OX5 1GB OXFORD, pp. 397–403.

[97] A.A. Shabana. *Dynamics of Multibody Systems*. Cambridge University Press, 1998.

[98] Hang Si. "Three Dimensional Boundary Conforming Delaunay Mesh Generation". PhD thesis. Institute of Mathematics, Technische Univerisität Berlin, 2008.

[99] A. Stern and B. Javidi. "Three-Dimensional Image Sensing, Visualization, and Processing Using Integral Imaging". In: *Proceedings of the IEEE* 94.3 (2006), pp. 591–606.

[100] W.R. Stevens. *TCP/IP Illustrated, Vol. 1: The Protocols*. Vol. 1. Addison-Wesley, 1993.

[101] I. Stroud. *Boundary Representation Modelling Techniques*. Springer, 2006.

[102] Ivan E. Sutherland, Robert F. Sproull, and Robert A. Schumacker. "A Characterization of Ten Hidden-Surface Algorithms". In: *ACM Computing Surveys* 6.1 (Mar. 1974), pp. 1–55.

[103] R. M. Taylor II et al. "VRPN: a device-independent, network-transparent VR peripheral system". In: *Proceedings of the ACM symposium on Virtual reality software and technology*. VRST '01. Baniff, Alberta, Canada: ACM, 2001, pp. 55–61.

[104] William C. Thibault and Bruce F. Naylor. "Set operations on polyhedra using binary space partitioning trees". In: *SIGGRAPH Computer Graphics* 21.4 (Aug. 1987), pp. 153–162.

[105] R.F. Tobler and S. Maierhofer. "A mesh data structure for rendering and subdivision". In: *14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2006, WSCG'2006 - In Co-operation with EUROGRAPHICS, Full Papers Proceedings*. 2006, pp. 157–162.

[106] R. Torres, V. Parra-Vega, and F.J. Ruiz-Sanchez. "Dynamic Haptic Training System for the Operation of an Excavator". In: 1^{st} *International Conference on Electrical and Electronics Engineering*. 2004.

[107] R. Torres, V. Parra-Vega, and F.J. Ruiz-Sanchez. "Integration of Force-Position Control and Haptic Interface Facilities for a Virtual Excavator Simulator". In: *International Conference on Advanced Robotics*. 2005.

[108] Z. Towarek. "Dynamics of a Single-Bucket Excavator on a Deformable Soil Foundation during the Digging of Ground". In: *International Journal of Mechanical Sciences* 45.6–7 (2003), pp. 1053–1076.

[109] Inc. USB Implementers Forum. *USB HID Information*. URL: `http://www.usb.org/developers/hidpage/`.

[110] Werner Verhelst and Marc Roelands. "Overlap-Add Technique Based on Waveform Similarity (WSOLA) for High Quality Time-Scale Modification of Speech". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing*. Vol. 2. 1993, pp. 554–557.

[111] A.D. Walker et al. "Head movements and simulator sickness generated by a virtual environment". In: *Aviation Space and Environmental Medicine* 81.10 (2010), pp. 929–934.

[112] M. Woof. "A Boom Market for Excavators". In: *Engineering and Mining Journal* 206.5 (2005), pp. 50–52.

[113] J. A. Zukas et al. *Impact dynamics*. New York: John Wiley and Sons, 1982.

[114] Y. H. Zweiri, L. D. Seneviratne, and K. Althoefer. "Modeling of a Closed-Chain Manipulators on an Excavator Vehicle". In: *Mathematical and Computer Modelling of Dynamical Systems* 12.4 (2006), pp. 329–345.