

## Department of Precision and Microsystems Engineering

### **RIGID-BODY SIMULATION WITH GAMING ENGINES: A study on the usability of gaming industry software for simulating engineering problems.**

J.H.H. (Jan Hein) de Jong

Report no : EM 2014.013  
Coach : dr. ir. P. Tiso  
Professor : prof. dr. ir. A. van Keulen  
Specialisation : Engineering Mechanics  
Type of report : Master Thesis



# **RIGID-BODY SIMULATION USING GAMING ENGINES**

**A study on the usability of gaming industry software  
for simulating engineering problems.**

MASTER OF SCIENCE THESIS

For the degree of Master of Science in Mechanical Engineering at Delft  
University of Technology

J.H.H. de Jong

June 12, 2014

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of  
Technology



The work in this thesis has been carried out in cooperation with the European Space Agency.



Copyright © Precision & Microsystems Engineering (PME)  
All rights reserved.

*PME - the Ultimate in  
Mechanical Engineering*



DELFT UNIVERSITY OF TECHNOLOGY  
DEPARTMENT OF  
PRECISION & MICROSYSTEMS ENGINEERING (PME)

The undersigned hereby certify that they have read and recommend to the Faculty of  
Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis  
entitled

RIGID-BODY SIMULATION  
USING GAMING ENGINES

by

J.H.H. DE JONG

in partial fulfillment of the requirements for the degree of  
MASTER OF SCIENCE MECHANICAL ENGINEERING

Dated: June 12, 2014

Supervisor(s):

---

ir. K. Wormnes

---

dr.ir. P. Tiso

Reader(s):

---

prof.dr.ir. A. van Keulen

---

dr.ir. P.T.L.M. van Woerkom

---

ir. M.V. van der Seijs



---

# Abstract

Rigid-body simulation, a sub-class of multi-body simulation, studies the movement of systems of interconnected undeformable bodies under the action of external forces. Through models based on the rigid-body assumption, many established software packages such as SIMPACK and MSC/Adams allow engineers to visualize and study the motion of mechanical systems. Examples in the aerospace industry where rigid-body simulations have been used are numerous and range from visualizing planetary rover behavior to studying separation dynamics for launch vehicles. Rigid-body simulations also have an increasingly important role to play in simulating orbital robotics.

The entertainment industry has been using rigid-body simulation techniques to create interactive environments for video games. The software packages developed for these applications, usually referred to as *gaming engines* or *physics engines* are able to simulate a large number of bodies on a normal desktop computer in near real-time. The big difference between the software specifically developed for engineering purposes and gaming engines is that the latter only need to provide plausible simulation, and thus have been developed with less stringent constraints on accuracy in mind. Nevertheless, early research by the European Space Agency on throw nets for capturing in-orbit debris has shown that these packages could prove to be useful for engineering purposes in some specific cases, due to their ease of use, high computational efficiency and open-source nature.

The main objective of this research is identifying the usability of gaming engines for engineering purposes. This can be split into four sub-objectives. The first objective is identifying and presenting the mathematical models that are common in gaming engines and assessing what their limitations are in terms of accuracy and stability. The second objective is identifying what would be the advantages of using gaming engines over traditional software packages. The third objective is indicating what applications would be well suited for simulation using gaming engines. And finally, the fourth objective is to propose a possible approach for creating a flexible rigid-body simulator specifically for engineering.

Most gaming engines use a method called constraint-based rigid-body simulation, based on the work by Stewart-Trinkle and Anitescu-Potra. This method uses a maximal coordinate formulation, describing joints and contacts through Cartesian coordinates and constraints.

Through a complementarity problem it computes the impulses that ensure that these constraints are satisfied. Friction is modeled using a linearized version of Coulomb's friction model. The resulting complementarity problem is typically solved using iterative solvers such as projected Gauss-Seidel, but other solvers are available, such as Lemke's algorithm. Time-integration of the equations of motion is typically done through a semi-implicit Euler method, with a fixed time-step size.

It is shown that the main source of inaccuracy and instability in gaming engines arise from the usage of the semi-implicit integration method and fixed time-step size. A proper selection of error reduction method and time-stepping algorithm is needed to prevent energy from being created in collisions. One should make sure that a proper solver and convergence criteria are used for solving the complementarity problem, in order to have accurate contact behavior. Finally, it should be noted that in general constraint-based simulation suffers from a lack of uniqueness in the reaction forces. However, this is inherent to the statically undetermined models arising from the rigid-body assumption.

It is shown that constraint-based simulation combined with a fixed time-stepping scheme is not very efficient in simulating stiff systems. However, the true power of these methods is revealed when simulating contact dynamics between a large number of rigid-bodies. The Stewart-Trinkle method performs several orders of magnitude faster than MSC/Adams.

Gaming engines can be very well suited for simulating applications involving a large number of rigid-bodies with contact dynamics at low cost. However, it is essential that the right combination of mathematical techniques are implemented to avoid the occurrence of unphysical behavior that might not be immediately visible to the user.

Numerous gaming engines exist worldwide, both open-source and closed-source, free and commercial. It is identified that Bullet is one of the most promising candidates to be adopted by the engineering world, since it is free and open-source, applies a proper selection of mathematical models and has a strong ongoing development.

---

# Table of Contents

<b>Preface</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1-1 Incentive . . . . .	2
1-2 Project goals . . . . .	3
1-3 Focus . . . . .	3
1-4 Notations . . . . .	4
<b>2 Literature survey</b>	<b>5</b>
2-1 Gaming engines . . . . .	5
2-1-1 Popular gaming engines . . . . .	6
2-1-2 Usage of gaming engines for engineering . . . . .	6
2-2 Constraint based simulation . . . . .	7
2-2-1 Stewart-Trinkle Method . . . . .	8
2-2-2 Constraint stabilization . . . . .	8
2-2-3 Semi-implicit Euler integration method . . . . .	9
2-2-4 Current research . . . . .	9
<b>I Theoretical analysis of gaming engines</b>	<b>11</b>
<b>3 Multi-body simulators</b>	<b>13</b>
3-1 Rigid-body simulators . . . . .	13
3-2 Modules . . . . .	14
3-2-1 Time-control . . . . .	14
3-2-2 Motion solver . . . . .	15
3-2-3 Constraint solver . . . . .	15
3-2-4 Collision solver . . . . .	15

3-3	Paradigms . . . . .	15
3-3-1	Penalty based . . . . .	16
3-3-2	Constraint based . . . . .	16
3-4	Formulations . . . . .	16
<b>4</b>	<b>Constraint-based simulation</b>	<b>19</b>
4-1	Newton-Euler equations . . . . .	19
4-2	Bilateral constraints - Joints . . . . .	20
4-3	Unilateral constraints - Contact . . . . .	22
4-4	Friction . . . . .	23
4-4-1	Linearized friction model . . . . .	24
4-4-2	Decoupled friction model . . . . .	27
4-5	Elastic collisions . . . . .	29
4-6	Constraint stabilization . . . . .	30
4-6-1	Baumgarte stabilization . . . . .	31
4-6-2	Post-stabilization . . . . .	34
4-6-3	Stabilization method comparison . . . . .	34
<b>5</b>	<b>The complementarity problem</b>	<b>37</b>
5-1	Linear and mixed-linear complementarity problems . . . . .	37
5-2	Linear complementarity problem solvers . . . . .	38
5-3	Treating a non-linear problem as a linear problem . . . . .	40
5-3-1	Decoupling the normal impulse and friction boundaries. . . . .	41
5-3-2	Exploiting the iterative nature of the PGS algorithm. . . . .	41
5-4	Solution existence and uniqueness . . . . .	42
<b>6</b>	<b>Bullet &amp; Blender</b>	<b>43</b>
6-1	Model identification . . . . .	43
6-1-1	Time-integration . . . . .	43
6-1-2	Friction model . . . . .	44
6-1-3	Constraint stabilization . . . . .	45
6-1-4	Complementarity problem solver . . . . .	47
6-2	Bullet & Blender algorithm structure . . . . .	47
6-2-1	Rendering loop . . . . .	48
6-2-2	Logic loop . . . . .	48
6-2-3	Physics loop . . . . .	49
6-2-4	Bullet algorithm overview . . . . .	49
6-3	Inaccuracies . . . . .	49
6-3-1	Semi-implicit Euler integration . . . . .	49
6-3-2	Collisions . . . . .	50
6-3-3	Joint stabilization . . . . .	52
6-3-4	Inertia tensor computation . . . . .	54
6-3-5	Solution convergence and uniqueness . . . . .	56

<b>II</b>	<b>Experimental verification</b>	<b>59</b>
<b>7</b>	<b>Elastic collisions</b>	<b>61</b>
7-1	Test setup . . . . .	61
7-2	Experimental and simulated results . . . . .	62
7-3	Comparison to established tools . . . . .	65
<b>8</b>	<b>String dynamics</b>	<b>67</b>
8-1	Test setup . . . . .	67
8-2	Model . . . . .	70
8-2-1	Spring forces . . . . .	70
8-2-2	Damping forces . . . . .	71
8-2-3	Slack . . . . .	71
8-2-4	Element properties . . . . .	72
8-2-5	Drag . . . . .	72
8-2-6	Implementation in Matlab, Blender & Adams . . . . .	73
8-3	Experimental & simulated results . . . . .	73
8-3-1	Experimental results . . . . .	74
8-3-2	Number of mass elements . . . . .	76
8-3-3	Comparison . . . . .	77
8-3-4	Other excitations . . . . .	77
8-4	Stability and convergence . . . . .	80
<b>9</b>	<b>Net behavior</b>	<b>83</b>
9-1	Setup and model . . . . .	83
9-2	Experiment versus simulation . . . . .	85
9-3	Sensitivity analysis . . . . .	86
<b>10</b>	<b>Computational efficiency</b>	<b>93</b>
10-1	Stiff systems . . . . .	93
10-2	Contact dynamics . . . . .	94
<b>11</b>	<b>Conclusions &amp; recommendations</b>	<b>97</b>
11-1	Conclusions . . . . .	97
11-2	Recommendations . . . . .	98
<b>A</b>	<b>Tracking</b>	<b>101</b>
A-1	Preprocessing . . . . .	101
A-1-1	Region of interest . . . . .	101
A-1-2	Gray-scale & Contrast . . . . .	101
A-1-3	Apply threshold . . . . .	102
A-2	Correlation . . . . .	102

<b>B</b>	<b>Semi-implicit Euler</b>	<b>105</b>
<b>C</b>	<b>Bullet &amp; Blender algorithm structure</b>	<b>107</b>
C-1	Blender . . . . .	108
C-2	Bullet . . . . .	109
	<b>List of References</b>	<b>111</b>
	<b>Glossary</b>	<b>115</b>
	List of Acronyms . . . . .	115
	List of Symbols . . . . .	115



---

# Preface

As cliché as it might sound, one of my first memories is actually visiting the European Space Agency's on-site museum, Space Expo, with my grandmother Lia and cousin Han. I was truly captivated by the images of astronauts bobbing around in their clumsy but oh so shiny white space suits, the amazing colors of the numerous pictures of constellations, planets and the earth and the technological achievements of the engineers that made it all possible.

Apart from space exploration, classical mechanics has captivated me also, ever since I learned the basic principles in high school. What attracted me the most, is that it presented challenging puzzles describing the motion of objects, with very tangible outcomes.

Needless to say, I was of course overjoyed to get a position to do this work at ESA, combining two fields of interest of me.

I would not have been able to finish this work on my own, and therefore I would hereby like to acknowledge the people that have helped me greatly during this project. First I would like to acknowledge the people at ESA for coming up with the project and facilitating, guiding and helping me with experiments, especially Kjetil Wormnes, Gianfranco Visentin, Tim Wiese, Robin Nelen, Carlos Crespo (Automation & Robotics section, ESTEC Noordwijk) and Gianluigi Baldesi (Mechanisms section, ESTEC Noordwijk).

Second I would like thank my supervisor from the university, Paolo Tiso (Department of Precision & Microsystems Engineering, Faculty of 3mE, Delft University of Technology). He has helped me in giving direction to my work, and he supported and motivated me when the going got tough.

I thank Erwin Coumans (Google, Bullet Physics) and Kenny Erleben (Department of Computerscience, University Copenhagen). Their work has been the starting point for my research, and I thank them for personally taking the time to answer my questions.

And last but certainly not least, I would like to thank my friends, family and girlfriend for supporting me through this sometimes challenging period. I thank my parents in particular, for providing me with everything I needed to get to this point in my life.

Delft, University of Technology  
June 12, 2014

J.H.H. de Jong



“Je n’ai fait celle-ci plus longue que parce que je n’ai pas eu le loisir de la faire plus courte.” (I only made this one longer because I did not have the leisure to make it shorter.)

— *Blaise Pascal, Lettres Provençales, Lettre XVI, 4 Dec. 1656*



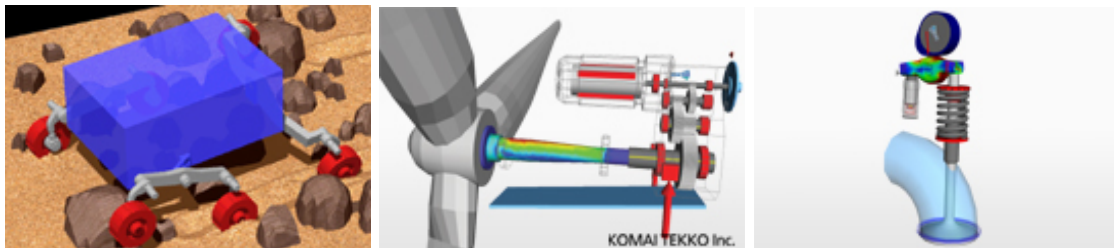
---

# Chapter 1

---

## Introduction

Multi-body simulation is the field of simulating dynamic systems of several interconnected or otherwise interacting rigid or flexible bodies. It enables engineers to visualize motion of complex 3D systems, and predict coupling forces and stresses, making it a valuable asset in designing and optimizing these types of systems. A short survey of the web results in numerous industry applications of these simulations. From planetary rovers to windmills and from guns to piston engines, the list is endless.



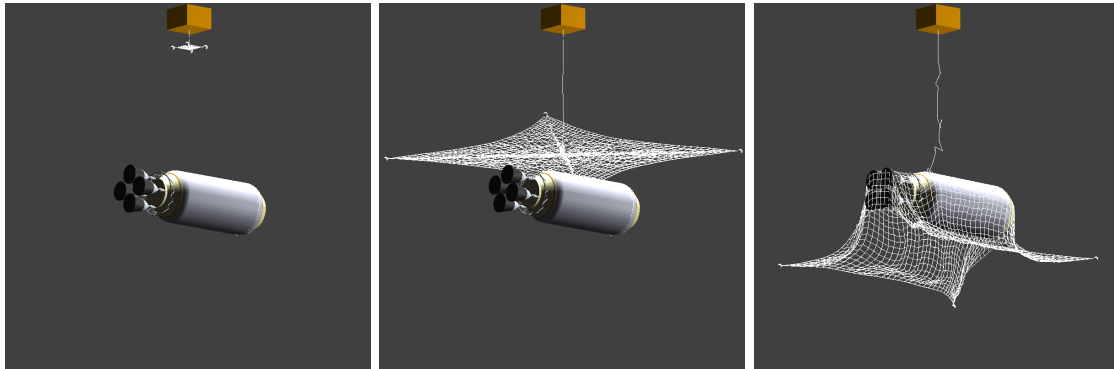
**Figure 1-1:** Examples of multi-body simulations[1].

However, engineers are not the only professionals using these type of simulations. The entertainment industry has also been using physics simulations very successfully in recent decades. Almost all computer games use at least some form of physics simulation these days, allowing its users to interact with a 3D environment that gives a sense of realism.

One key element of these gaming industry physics engines is that they are required to run on low-performance household PC's. Therefore the mighty gaming industry has spent considerable amounts of resources on improving the computational efficiency of these engines, allowing them to simulate amazing amounts of interacting bodies in real-time on a simple system.

Recently, the European Space Agency has started investigating the usability of gaming engines to perform upstream multi-body simulations. A driver for this was the need for simulations of throw nets to capture malfunctioning satellites, and re-orbit them. Nets can be modeled as very large lumped-mass models. However, it turned out that traditional multi-body simulation

software did not match ESA's needs in terms of speed and ease of use. Therefore ESA resorted to using the 3D modeling package Blender, and its built-in gaming engine, Bullet. An example of these simulations is shown in Figure 1-2.



**Figure 1-2:** A simulation of a throw-net capturing a burned-up rocket stage[2].

However, the major difference between a scientific multi-body simulator, and a gaming physics engine, is that the latter is not required to generate realistic simulations, only realistically *looking* simulations. This philosophy has a big impact on the design of something as complex as multi-body simulator. Usually, there is a trade-off between accuracy, and computational expense. Where the gaming industry programmer might be inclined to choose speed over accuracy, an engineer will have a higher requirement on accuracy, or at least wants to be able to identify the accuracy of the simulation, as not to base judgments on faulty simulations.

Furthermore, where someone playing a video game is solely interested in the visible motion of objects, engineers are typically interested in internal loads of the system. This allows them to compute stresses, and set design criteria to prevent failure of mechanisms and structures.

## 1-1 Incentive

There are several incentives that justify investigating the use of gaming engines for simulating a certain class of engineering problems. First, initial research with Bullet has shown very high performance in terms of complexity of systems that could be simulated on a regular PC in limited amount of time.

Second, a lot of gaming software is open-source, and therefore free to use. As a comparison, licenses for traditional multi-body simulators can cost up to several tens of thousands of euros per core.

Finally, modeling systems using Blender has proved to be very easy and fast. The interface is intuitive, and many tutorials are available on-line, which allows new users to quickly get acquainted with the program. On top of that, Blender is also a scriptable environment. This enables the user to customize many functions, program the modeling, and add complex elements such as controllers, sensors and actuators. Therefore ease of use is not at the expense of functionality.

## 1-2 Project goals

The goals of this project can be summarized into four main topics.

- Identify the underlying mathematical methods of gaming engines and assess their accuracy.
- Determine what would be the advantages of using a gaming engines over traditional software.
- Define what applications would be well suited to be simulated with gaming engines.
- Propose a possible implementation for an engineering oriented simulator based on gaming engines.

Before basing decisions on simulations performed with gaming engines, we should know what the reliability of these simulations is. Is accuracy traded-off to reduce computational expense, and if so where? In what situations can we expect the simulations to behave physically correct, and to what extent? The best way to answer these questions is to gain a deep understanding the of mathematical models at the heart of the simulator.

This understanding, together with computational experiments will allow us to determine and confirm the suspected advantages gaming engines can have over traditional tools (i.e. speed, flexibility, ease-of-use). We can also define in which cases these advantages would outweigh possible disadvantages.

Finally, if gaming engines indeed prove useful in some cases, we can define what would be the best way towards a purpose built engineering simulator. Is it best to build on an existing framework and if so which? How can we employ the potential of gaming engines to best fit the needs of engineers?

## 1-3 Focus

The focus on this work is mainly on applied mechanics and less on programming or IT related issues. During the first phase of the project it was found that the theory on the mathematical methods used in gaming engines is spread over numerous publications from the past thirty years. Therefore the aim of this work is to provide a clear and easy to understand, yet complete introduction to the most popular methods used today, allowing a reader who is new to the field of rigid-body simulators to understand all the fundamental components of a gaming engines and how they relate to one another. All research related to this purpose is bundled in Part I of this report.

Furthermore it is the author's belief that the best way to identify the reliability and usability of simulations is to look at theory and computer experiments, and in addition to that to step out of the digital world and compare real systems to simulations. Therefore a large portion this research has been allocated to comparing simulations to actual experimental data. This work is presented in Part II of this report.

Although not the initial intention, this research has stayed relatively close to the original application that started this project (i.e. simulating throw-nets). However, the physics involved in the presented problems is of course applicable to a wide range of problems.

## 1-4 Notations

Throughout this work, a vector will be noted as a bold lower-case letter (i.e.  $\mathbf{s}$ ) and matrices are indicated as italic bold upper-case letters (i.e.  $\mathbf{A}$ ). A scalar is noted as an italic lower-case letter (i.e.  $m$ ).



---

## Chapter 2

---

# Literature survey

The first part of this chapter aims to give an overview of the most used gaming engines, and explains the main difference between them and conventional engineering simulators. Also we look at previous attempts to use these tools for engineering purposes. The second part gives an overview of how the most important mathematical methods elaborated in this thesis came to be.

### 2-1 Gaming engines

A simple search of the web results in a long list of physics simulators. One finds commercial engineering tools like MSC/Adams, tools aimed at computer games like Bullet, and testbeds for new simulation techniques such as dVC3d[3, 4, 5].

Engineering software have a lot of functionality, including frequency-domain analyses, finite element modeling of compliant structures, optimization of cost functions, and so on. The problems that need to be solved in entertainment products are of a different nature. One can think of real-time simulations of rag dolls crashing into each other, a brick wall falling, or cars crashing into each other. Gaming engines therefore have a strong focus on contact dynamics of large numbers of rigid-bodies in the time domain.

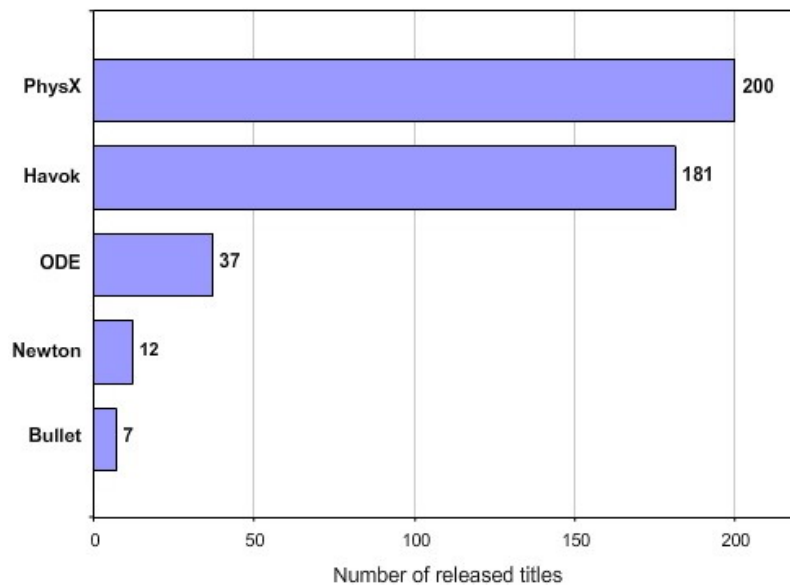


**Figure 2-1:** A typical simulation made with Blender & Bullet for entertainment purposes[6].

### 2-1-1 Popular gaming engines

A good starting point for charting the field of available gaming engines is to determine which gaming engines have been adopted the most by their target group (i.e. video game developers). Figure 2-2 shows the result of a survey performed by electronics company Nvidia.

Although possibly not the most unbiased survey imaginable, it shows what are considered to be the main players in the field. Three out of five of these engines are backed by large budgets from electronics companies. PhysX is developed at Nvidia, Havok at Intel and Bullet at Advanced Micro Devices (AMD)<sup>1</sup>.



**Figure 2-2:** Comparison of most used gaming engines by number of released commercial video-games from 2006 to 2009[7].

The comparison shown in Table 2-1 teaches us two things. First, we see that all popular gaming engines use the so-called Stewart-Trinkle constraint based method for solving contact problems. As a comparison, both Adams and SIMPACK use different, penalty based methods for handling contacts[3, 8]. Second, we see that Bullet is a strong candidate to use as a basis for exploring the possibilities of gaming engines. Due to its open-source nature the software is more transparent, allowing us to more easily define and analyze what steps the software takes during a simulation. Also, Bullet has a strong ongoing development, ensuring that it uses the most modern techniques developed in the rapidly evolving world of rigid-body simulation.

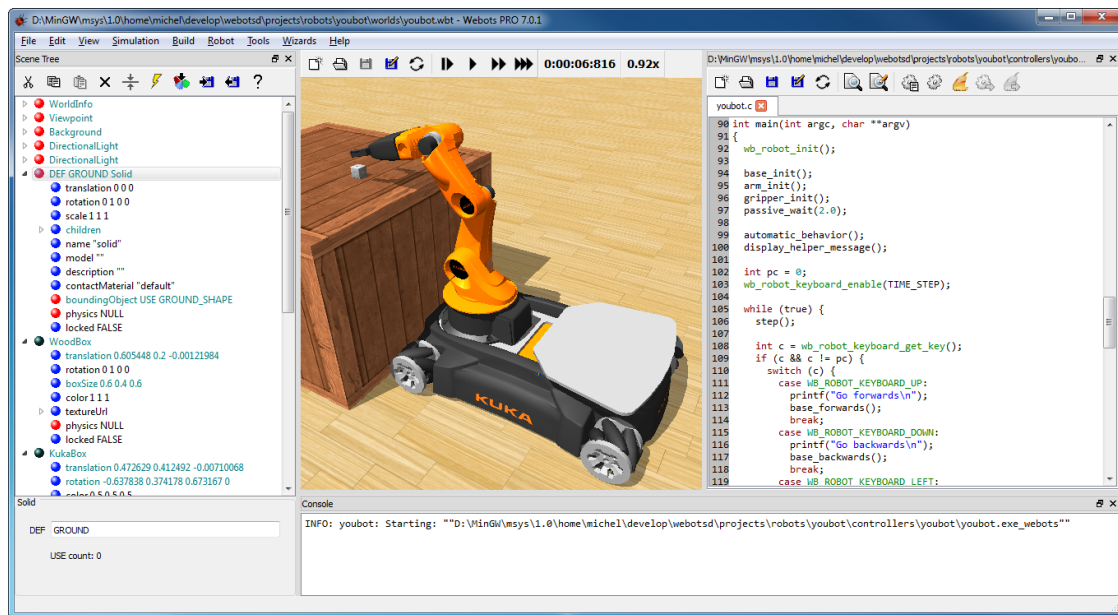
### 2-1-2 Usage of gaming engines for engineering

Gaming engines have been used in engineering before, mainly in robotics. There exist several tools for rapid-prototyping of robotic systems, which house one of the gaming engines mentioned above[13, 14]. These tools experiment with robot designs and algorithms in a virtual

<sup>1</sup>During the writing of this thesis, the creator of Bullet Physics stopped working at AMD and was hired at Google

**Table 2-1:** Overview of most used gaming engines

Name	Contact model	Distribution	Development
PhysX SDK [9]	Stewart-Trinkle	Closed-source	Ongoing
Havok Physics [10]	Stewart-Trinkle	Closed-source	Ongoing
Open Dynamics Engine [11]	Stewart-Trinkle	Open-source	Suspended
Newton [12]	Stewart-Trinkle	Closed-source	Ongoing
Bullet [4]	Stewart-Trinkle	Open-source	Ongoing

**Figure 2-3:** A screenshot from the robotics development suite Webots[14].

environment, eliminating issues involved with real-world experiments such as short battery life, hardware failures and unexpected and dangerous behavior. In some cases the simulator can run faster than real-life, further speeding up the process and enabling Monte-Carlo analyses to be performed. ESA also co-developed a testbed for planetary rovers exploring the surface of Mars, called 3Drov[15].

A probable future application of fast simulation techniques involving contact dynamics is in control of complex robots. Imagine a robot having to perform tasks in a real world environment, for instance cleaning objects or traversing a burning building. The robot will have to be able to predict the outcome of its actions into the future just like humans do (if I move this brick, this wall will fall). This would require a very fast, generic, built-in simulator[16].

## 2-2 Constraint based simulation

As we saw in Section 2-1, most gaming engines are based on the Stewart-Trinkle method. This is a so called constraint based method for solving contact problems. This approach is often referred to as constraint based simulation. In this method movement of bodies is restricted

to prevent penetration through unilateral constraints. This first approach to solving contact in a constraint based way was outlined in by P. Lötstedt in 1979 and a refined version was published in 1984 [17, 18]. In this method, a Lagrange multiplier approach is taken, and the simulation is performed by solving a quadratic programming optimization problem at every time-step. A different algorithm was proposed by David Baraff in 1994 [19]. This method formulated the problem in the form of a Linear Complementarity Problem (LCP). The main advantage of this approach is that it could be implemented without the need for large-scale optimization software packages, and that it was considerably faster. This publication formed the basis for what would later be known as the Stewart-Trinkle method.

### 2-2-1 Stewart-Trinkle Method

Baraff formulated his method on a forces and acceleration level. In the event of an impact between two rigid bodies they experience discontinuous velocities which requires infinite reaction forces. A method formulated on a force-acceleration cannot handle these events, and velocities need to be altered separately based on some impulse-momentum law [16].

A different approach was proposed by D. Stewart and J. Trinkle in 1996 [20]. This method, commonly referred to as the Stewart-Trinkle method, formulates the contact problem on a impulse-momentum level, earning it the name *'impulse based simulation'*. This is achieved by effectively integrating the forces and accelerations over the length of a time-step. This eliminates the need to explicitly resolve impulse forces arising from rigid-body collisions. Note that a method can be impulse based and constraint based at the same time, the respective terminology refers to different parts of the formulation.

The original Stewart-Trinkle method sometimes suffered from the lack of a solution. Therefore the method was adapted by M. Anitescu and F.A. Potra to include a non-penetration constraint on a velocity level [21]. In the same publication it was shown that this formulation always has a solution.

A modular approach for building an simulator based on the Stewart-Trinkle method was proposed by K. Erleben in his thesis and book [22, 23]. These extensive works propose a generic notation combining all several components needed for applications of rigid-body simulation (i.e. joints, motors, joint limits). The notation is based on work done by J. Sauer and E. Schömer [24].

### 2-2-2 Constraint stabilization

The Stewart-Trinkle method typically uses a maximal coordinate formulation, where the total number of degrees of freedom is not reduced by a constraint, but constraints are enforced on the possible solutions. Combining this with numerical integration of the equations of motion leads to drifting of the constraints. A well known method for stabilizing of constraints is the Baumgarte error reduction method, published in 1972 [25]. M. Cline and D. Pai proposed a more accurate error reduction method in the complementarity framework [26].

### 2-2-3 Semi-implicit Euler integration method

As described in appendix B, the semi-implicit Euler integration method is a combination of the basic explicit and implicit Euler integration methods. The difference between an implementation of an explicit and a semi-implicit Euler method is a matter of switching two lines in a computer script, and therefore it can be assumed that its discovery cannot be attributed to one single person. However, the first publication analyzing its behavior was by A. Cromer in 1981, and therefore the method is also referred to as the Euler-Cromer method [27]. A more thorough investigation of its behavior compared to other so-called symplectic integrators was presented in 2005 by D. Donnelly and E. Rogers [28].

### 2-2-4 Current research

Current research on constraint based simulation focuses on utilizing the parallel computing power of a Graphics Processing Unit (GPU) for collision detection and solving complementarity problems [16, 29]. Other research focuses on more efficient algorithms for solving complementarity problems [30, 31].



## **Part I**

# **Theoretical analysis of gaming engines**





# Multi-body simulators

Multi-body simulators have a wide range of applications [16]. In the engineering world they are used for several purposes. The first and most obvious is speeding up design processes. Engineers can simulate designs in stead of building prototypes. One can even apply some automated optimization process to find ideal values for certain parameters.

Another engineering application is in virtual-reality simulators. These can be used to train professionals like crane operators and captains of tow-vessels. They can also be used to simulate real people interacting with a newly designed environment, for instance a fork-lift driver in a warehouse, enabling designers to verify the ergonomics of the environment.

Finally, in the near future multi-body simulators can also play a vital role in robot control systems. Robots are designed to perform tasks that involve predicting and judging interaction with its surrounding. Think of moving obstacles and grasping moving objects. Just like a human sub-conscious mind predicts movements, so will the robot have to predict the effects of its actions.

In the entertainment industry, multi-body simulators have mainly been developed for computer games. They allow users to interact with a world that behaves in a plausible way. Realism has reached a high enough level when a viewer does not notice the inaccuracies.

### 3-1 Rigid-body simulators

In the real world, a ball bouncing on a surface will deform, storing all kinetic energy as elastic energy, and later release this energy again in the form of kinetic energy. One way to simulate this is through a compliant body model such as finite element analysis. In such a method the internal deformation of the body is modeled, at a high computational cost.

A different subclass of multi-body simulators is a rigid-body simulators. In such a simulator all bodies are assumed to be non-deformable, and contact interactions are calculated based on impulse-momentum laws. Often contact between two surfaces is simplified to a set of contact points with reaction forces acting on them. As a consequence of this assumption, any internal stresses are not calculated.

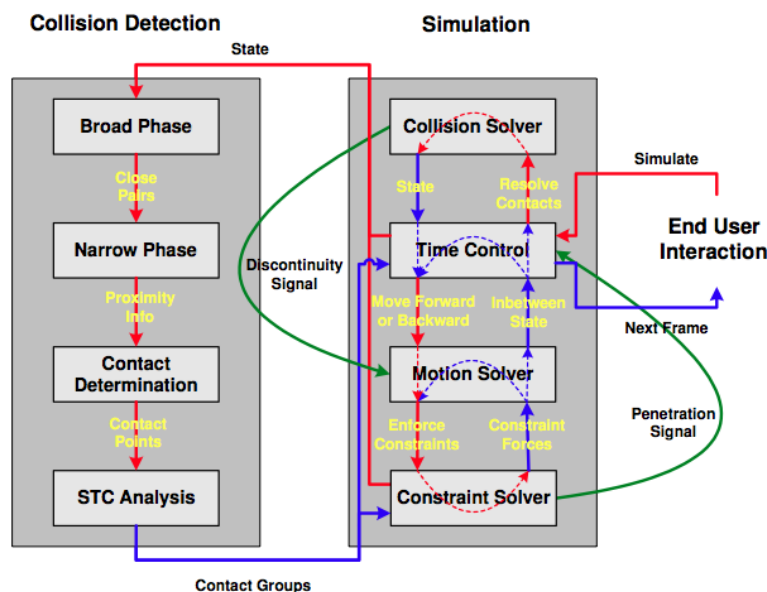
## 3-2 Modules

A clear way of describing all the components of a rigid-body simulator is the modular approach proposed by K. Erleben [22]. He states that on a high level, a simulator can be divided into two components: a simulation component, and a collision detection component.

The collision detection checks which objects are touching, and returns a set of contact points that best approximate the contact. Our focus is on the simulation component, which can be subdivided into four modules.

- Time-control
- Motion solver
- Constraint solver
- Collision solver

Figure 3-1 illustrates these modules and their interactions.



**Figure 3-1:** A modular overview of the components in a rigid-body simulator [22]

### 3-2-1 Time-control

The time-control module is really the heart of the simulator. It controls when other modules are invoked. Simulation is started when a certain set of initial conditions is fed to the time-control, together with a desired simulation end time. It invokes the other modules several times, depending on the desired physics accuracy, until the end time is reached. At this point, the time-control module returns the updated world. This loop is referred to as the outer loop.

In the case of computer game, the time interval of one outer loop is usually in the same order of magnitude as the rate at which the screen is refreshed. This allows the user to interact with the world between each iteration of the outer loop.

An important characteristic of the time-control module is how it defines its time-step. One way is to have a fixed time-step, resulting in small penetrations between colliding bodies. Another method is backtracking, where as soon as a collision is detected, the simulation is rewound to the exact instant of the collision, preventing penetration. Bullet uses a fixed time-stepping algorithm.

### 3-2-2 Motion solver

The state of a system at any point in time can be described as the position, orientation and velocities of all the bodies in the system. The change of this state is described by the Newton-Euler equations, linking accelerations to forces acting on the system. The set of ordinary differential equations that are described by these Newton-Euler equations cannot be solved analytically, and are therefore solved numerically. The motion solver receives the state of the system and all the forces acting on the system over a certain interval, and returns the state at the end of that interval.

### 3-2-3 Constraint solver

Two types of constraints act on any system of rigid-bodies.

- Bilateral constraints ( $=$ )
- Unilateral constraints ( $\geq$ )

Bilateral constraints are used to describe joints, unilateral constraints to describe contacts. The constraint solver computes all joint and contact forces over a certain interval, and returns them to the motion solver.

### 3-2-4 Collision solver

The term collision refers to a situation where two bodies that at a previous time-step were not yet in contact are suddenly touching or penetrating. This is different from a normal contact in the sense that the involved bodies can have some motion relative to each other. Depending on the elastic properties of the bodies, this relative motion needs to either be reduced to zero (inelastic collision), or even be opposite the initial relative velocity (elastic collision). This can be handled by the a separate collision solver, or included in the constraint solver.

## 3-3 Paradigms

Since rigid-bodies are un-deformable, we need a different way of describing the forces that act between two bodies in contact. There are basically two ways to handle this, either through some penalty function, or through constraints [32].

### 3-3-1 Penalty based

In penalty based simulation, contact forces or contact impulses normal to the contact plane (i.e.  $p_c$ ) are described by penalty functions, usually based on penetration depth (i.e.  $\delta s$ ) and velocity (i.e.  $\delta \dot{s}$ ). Physically this can be interpreted as adding a spring-damper between the bodies at the point of contact that can only deliver a compressive (i.e. positive and real,  $\mathbb{R}^+$ ) force.

$$p_c(\delta s, \delta \dot{s}) \in \mathbb{R}^+ \quad (3-1)$$

Penalty based simulation is simple to implement, and compliant contacts can be approximated quite easily [16].

However, it also suddenly increases stiffness of the system when there is contact, requiring a much smaller time-step in order for the system to remain stable. If the time-step size is adjusted to the stiffness of the system, this leads to unpredictable computation time per simulated interval (i.e. one second of simulation can cost one-tenth or two seconds to compute, depending on the number of contacts).

### 3-3-2 Constraint based

Another fundamentally different way of approaching the problem is constraint-based. In this approach, the reaction forces are defined such that they enforce a constraint on either the position or the velocity of the objects, preventing (further) penetration.

$$\text{Find } p_c \in \mathbb{R}^+ \text{ such that } \delta s \geq 0 \vee \delta \dot{s} \geq 0 \quad (3-2)$$

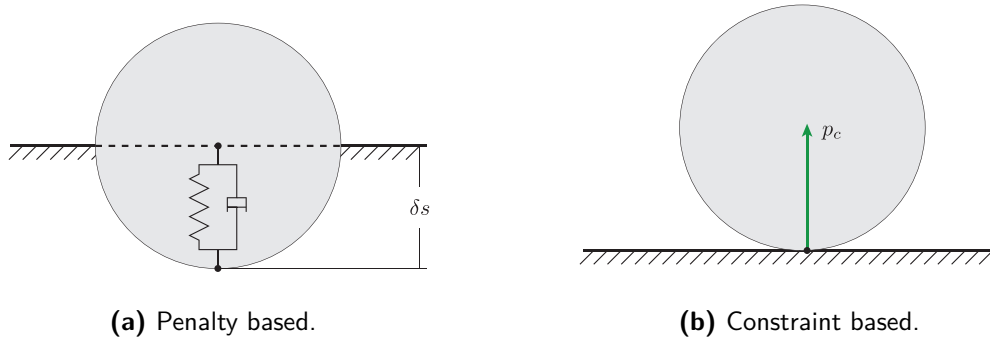
The disadvantage of this method is that a complex system of equations needs to be solved, and that either the position or velocity of the objects at the next time-step needs to be known explicitly in terms of the reaction forces. The advantage of this method is that the time-step size does not need to be reduced when there is an active contact.

The difference between penalty based and constraint based simulation is illustrated in Figure 3-2.

## 3-4 Formulations

If a system incorporates joints, the number of degrees of freedom of the system is reduced. Think of a two dimensional point mass. We would need a  $x$ - and  $y$ -coordinate to describe the position of the mass. If it is constrained by a hinge, we can describe its position in space using the two coordinates, and using a Lagrange multiplier method enforce the constraint on the position. This method is referred to as the *maximal coordinate formulation*.

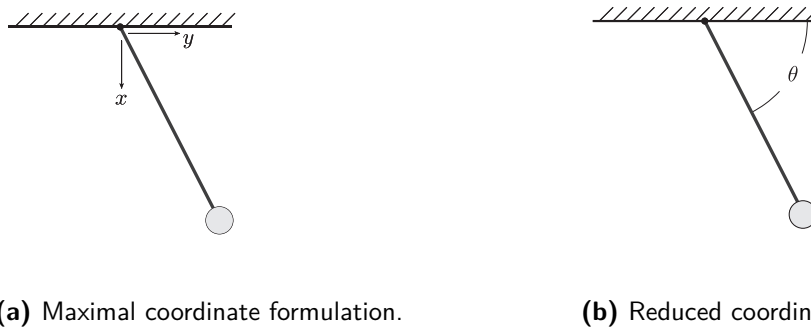
We can also eliminate a degree of freedom through the constraint equations, and choose some generalized set of constraints. In the case of our point mass on a hinge, we could describe



**Figure 3-2:** Conceptual representation of the two fundamental paradigms used to handle rigid-body contact.

its position in space through the hinge angle  $\theta$ . This method is called the *reduced coordinate formulation*. A popular reduced coordinate method is Featherstone's algorithm [33].

Maximal coordinate formulations suffer from constraint drifting and need to be stabilized. However, the formulation is usually more clear and easy to implement. The Stewart-Trinkle uses a maximal coordinate formulation [16].



**Figure 3-3:** Conceptual representation of the two fundamental formulation paradigms used to describe interconnected rigid-bodies.



# Constraint-based simulation

This chapter describes the mathematics behind the Stewart-Trinkle method. To improve readability some citations regarding this method are omitted in this chapter. The reader is referred to Chapter 2 for a description of the development of this method.

We also adopt a simple notation, and focus on the parts where inaccuracies occur in Bullet. All equations in this report are written for one rigid-body interacting with a static world. If one is interested in a more extensive notation that includes multiple bodies, the reader is referred to K. Erleben's book on the subject [23].

The reader is expected to have basic understanding of quaternions. It is enough to know it is a way to describe orientations without suffering from a situation called 'Gimbal lock'. For a quick description of how quaternions can be applied in the Stewart-Trinkle method, the reader is referred to the work by Bender [16]. For a more detailed description of quaternions Hanson's work is a good starting point [34].

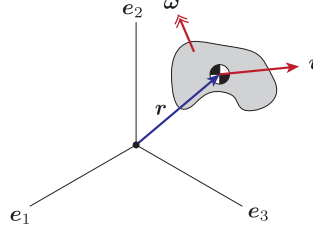
### 4-1 Newton-Euler equations

The state of a rigid-body, for example the potato shown in Figure 4-1 can be fully described by its position, orientation and their velocities. We adopt the vector  $\mathbf{r} \in \mathbb{R}^3$  to describe the position of the body in space, and the unit quaternion  $\mathbf{q} \in \mathbb{R}^4$  describing the orientation of the body. We can define a generalized position and orientation vector  $\mathbf{s} \in \mathbb{R}^7$ , such that

$$\mathbf{s} = \begin{bmatrix} \mathbf{r} \\ \mathbf{q} \end{bmatrix}. \quad (4-1)$$

We also adopt the vector  $\mathbf{v} \in \mathbb{R}^3$  for the translational velocity, and vector  $\boldsymbol{\omega} \in \mathbb{R}^3$  for the rotational velocity, defined in the body frame of reference. We can define a generalized velocity vector  $\mathbf{u} \in \mathbb{R}^6$ , such that

$$\mathbf{u} = \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix}. \quad (4-2)$$



**Figure 4-1:** Illustration of coordinate system. Linear position  $\mathbf{r}$  is concatenated with the unit quaternion to form the generalized position and orientation vector  $\mathbf{s}$ . The linear velocity  $\mathbf{v}$  and rotational velocity  $\boldsymbol{\omega}$  are concatenated to form the generalized velocity vector  $\mathbf{q}$ .

This allows us to write the Newton-Euler equations in a clear matrix form, yielding

$$\mathbf{M}\dot{\mathbf{u}} = \mathbf{f}_{\text{contact}}(\mathbf{s}, \mathbf{u}) + \mathbf{f}_{\text{ext}}(t^n, \mathbf{s}^n, \mathbf{u}^n), \quad (4-3a)$$

$$\dot{\mathbf{s}} = \mathbf{S}(\mathbf{s})\mathbf{u}, \quad (4-3b)$$

where  $\mathbf{M}$  is the generalized mass-matrix containing the mass of the object and the inertia tensor, and  $\mathbf{S}(\mathbf{s})$  is the kinematic map, mapping the rotational velocity into a changes in the quaternion. Note that the kinematic map is dependent on the orientation of the system. Vector  $\mathbf{f}_{\text{contact}}(\mathbf{s}, \mathbf{u})$  contains all forces resulting from contact between bodies and  $\mathbf{f}_{\text{ext}}(t, \mathbf{s}, \mathbf{u})$  contains the Coriolis forces and any external forces acting on the body (e.g. springs, dampers, gravity, controlled actuators).

This system is in most cases impossible to solve analytically. However, they can be solved using a numerical time-stepping scheme. There are many integration schemes, ranging from very basic ones (e.g. explicit Euler) to more stable and accurate methods (e.g. Newmark, fourth order Runge-Kutta). We will see that the Stewart-Trinkle method requires the velocities in the next time-step ( $\mathbf{v}^{n+1}$ ) to be explicitly dependent on the contact forces. Therefore it is often combined with an integration method called *semi-implicit Euler*, *symplectic Euler* or *Euler-Cromer* (see Appendix B). This yields

$$\mathbf{u}^{n+1} = \mathbf{u}^n + h\mathbf{M}^{-1}(\mathbf{f}_{\text{contact}}(\mathbf{s}, \mathbf{u}) + \mathbf{f}_{\text{ext}}(t^n, \mathbf{s}^n, \mathbf{u}^n)), \quad (4-4a)$$

$$\mathbf{s}^{n+1} = \mathbf{s}^n + h\mathbf{S}(\mathbf{s})\mathbf{u}^{n+1}, \quad (4-4b)$$

where  $h$  represents some arbitrary time-step size. The difference between this scheme and explicit Euler, is that position and orientation are updated based on the velocity at step  $n+1$ , whereas in explicit Euler these would be updated based on the velocity at step  $n$ .

For certain set of initial conditions and known forces acting on the body we can now approximate its unconstrained motion.

## 4-2 Bilateral constraints - Joints

If our body is connected to the fixed world through a joint, it cannot take any arbitrary position or orientation in space any more. It is in fact constrained to some kinematically



admissible subspace. A joint can be described by a number of bilateral, or equality constraint equations. Every bilateral constraint reduces the number of degrees of freedom by one. For simplicity we will only treat holonomic constraints. We denote any bilateral constraint as  $\phi(\mathbf{s})$ , and any combination of bilateral constraints can be described as

$$\phi(\mathbf{s}) = \begin{bmatrix} \phi_1(\mathbf{s}) \\ \vdots \\ \phi_m(\mathbf{s}) \end{bmatrix} = 0, \quad (4-5)$$

where  $m$  is the number of degrees of freedom that the system loses due to the joint. We can write our constraints explicitly for velocities by taking the first order Taylor approximation of the constraint equations.

$$\phi(\mathbf{s}^{n+1}) \approx \phi(\mathbf{s}^n) + \frac{\partial \phi(\mathbf{s})}{\partial \mathbf{s}}(\mathbf{s}^{n+1} - \mathbf{s}^n) = 0 \quad (4-6)$$

$$(4-7)$$

Rewriting and plugging the position update from the numerical integration (Eq. (4-4a)) yields

$$\phi(\mathbf{s}^{n+1}) \approx \phi(\mathbf{s}^n) + h \frac{\partial \phi(\mathbf{s})}{\partial \mathbf{s}} \mathbf{S}(\mathbf{s}) \mathbf{u}^{n+1} = 0. \quad (4-8)$$

The Jacobian  $\mathbf{J}_e$  of the equality constraints is evaluated at step  $n$  and is defined as

$$\mathbf{J}_e^n = \left. \frac{\partial \phi(\mathbf{s})}{\partial \mathbf{s}} \right|_n \mathbf{S}(\mathbf{s}^n). \quad (4-9)$$

If we assume that our constraint is satisfied at step  $n$ , we can simplify Eq. (4-8) as

$$\mathbf{J}_e^n \mathbf{u}^{n+1} = 0. \quad (4-10)$$

Because all Jacobians are always evaluated at  $n$ , we will improve readability by omitting the superscript  $n$  in the following.

We are interested in the reaction force  $\mathbf{f}_e$ , necessary to satisfy the bilateral constraints at step  $n+1$ . However, in our integration scheme (Eq. (4-4a)) we would integrate this force over the time interval (Eq. (4-4a)), obtaining an impulse. This can be written as

$$\int_n^{n+1} \mathbf{f}_e dt \approx h \mathbf{f}_e^n = \mathbf{p}_e. \quad (4-11)$$

Adopting a cleaner notation where we get straight to the impulse, the impulse needed to satisfy the constraints can be written using  $m$  Lagrange multipliers  $\lambda_e$  such that

$$\mathbf{p}_e = \mathbf{J}_e^T \lambda_e, \quad (4-12)$$

where the transposed Jacobian matrix represents a vector in the constrained direction. The Lagrange multiplier  $\lambda_e$  is related to the magnitude of the constraint impulse, but since the magnitude of the Jacobian  $\|\mathbf{J}_e^T\|$  is not necessarily equal to one,  $\lambda_e$  does not exactly represent the magnitude of the contact impulse.

Note that  $\mathbf{p}_e$  represents the contact impulse applied over the time-interval  $h$ . Therefore it makes no sense to associate it with a point in time (i.e.  $n$  or  $n+1$ ).

Combining this with the velocity update of the integration scheme, we can find the velocities at  $n + 1$  and the impulses required to satisfy the equality constraints,

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}_e^T \\ \mathbf{J}_e & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ \lambda_e \end{bmatrix} - \begin{bmatrix} \mathbf{M}\mathbf{u}^n + h\mathbf{f}_{\text{ext}}^n \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \quad (4-13)$$

Solving this and plugging the update velocities in the position update of the integration scheme (Eq. (4-4b)) will yield an approximation of the constrained motion of our body.

### 4-3 Unilateral constraints - Contact

If our body encounters another object, we need to constrain it in such a way that it does not penetrate the other object. We refer to such a constraint as a non-penetration, inequality or unilateral constraint. We denote a unilateral constraint as  $\psi(\mathbf{s})$ , and any set of unilateral constraints can be described as

$$\boldsymbol{\psi}(\mathbf{s}) = \begin{bmatrix} \psi_1(\mathbf{s}) \\ \vdots \\ \psi_k(\mathbf{s}) \end{bmatrix} \geq 0, \quad (4-14)$$

where  $k$  is the number of contact points the body has with other objects. Again, we can approximate the constraint by taking a Taylor approximation, and plugging the velocity update to obtain

$$\boldsymbol{\psi}(\mathbf{s}^{n+1}) \approx \boldsymbol{\psi}(\mathbf{s}^n) + \frac{\partial \boldsymbol{\psi}(\mathbf{s})}{\partial \mathbf{s}} (\mathbf{s}^{n+1} - \mathbf{s}^n) \geq 0, \quad (4-15a)$$

$$\approx \boldsymbol{\psi}(\mathbf{s}^n) + h \frac{\partial \boldsymbol{\psi}(\mathbf{s})}{\partial \mathbf{s}} \mathbf{S}(\mathbf{s}) \mathbf{u}^{n+1} \geq 0. \quad (4-15b)$$

The Jacobian  $\mathbf{J}_c$  of the inequality constraints is evaluated at step  $n$  and is defined as

$$\mathbf{J}_c^n = \left. \frac{\partial \boldsymbol{\psi}(\mathbf{s})}{\partial \mathbf{s}} \right|_n \mathbf{S}(\mathbf{s}^n). \quad (4-16)$$

Again we will omit the superscript  $n$  for the Jacobian to improve readability. We assume that all active contacts are touching at time  $n$ , and therefore the value of the constraint at step  $n$  is zero. We will later see that this approximation is very crude, but for now it allows us to simplify the constraint to

$$\mathbf{J}_c \mathbf{u}^{n+1} \geq 0. \quad (4-17)$$

We define the impulse  $\mathbf{p}_c$  needed to satisfy the non-penetration at step  $n + 1$  as

$$\mathbf{p}_c = \mathbf{J}_c^T \lambda_c. \quad (4-18)$$

However, we need to consider two other characteristics of the contact impulse. First, the contact impulse can only push on the body, never pull. Therefore  $\lambda_c$  can only be larger than zero. Second, if the relative velocity at the contact point is positive, the bodies are separating over the course of  $h$ . This means that they cannot exert a contact impulse on each other.

So either the relative velocity is zero, or the force is zero. These two characteristics can be written as a so-called complementarity condition. We write

$$\mathbf{J}_c \mathbf{u}^{n+1} \geq 0 \quad , \quad \lambda_c \geq 0 \quad , \quad a^T \lambda_c = 0, \quad (4-19)$$

where  $a$  is the relative velocity defined as

$$a = \mathbf{J}_c \mathbf{u}^{n+1}. \quad (4-20)$$

Now we can include the non-penetration conditions to our system.

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}_e^T & -\mathbf{J}_c^T \\ \mathbf{J}_e & 0 & 0 \\ \mathbf{J}_c & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{u}^{n+1} \\ \lambda_e \\ \lambda_c \end{bmatrix} - \begin{bmatrix} \mathbf{M} \mathbf{u}^n + h \mathbf{f}_{\text{ext}} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ a \end{bmatrix} \quad (4-21)$$

$$a \geq 0 \quad , \quad \lambda_c \geq 0 \quad , \quad a^T \lambda_c = 0$$

The form of this system is called a Mixed Linear Complementarity Problem (MLCP). It is mixed because it contains both equality conditions as well as inequality conditions. More on these types of systems can be found in the work by R. Cottle [35].

## 4-4 Friction

At the contact point the body can also experience friction. We model this using Coulomb's friction model, which has two 'modes'.

1. There is no relative velocity tangential to the contact plane at the contact point, i.e. the body is **rolling**.<sup>1</sup>
2. There is a relative velocity tangential to the contact plane, i.e. the body is **sliding**.

If the body is rolling then the magnitude of the friction force  $f_f$  is defined as

$$f_f \leq \mu f_c, \quad (4-22)$$

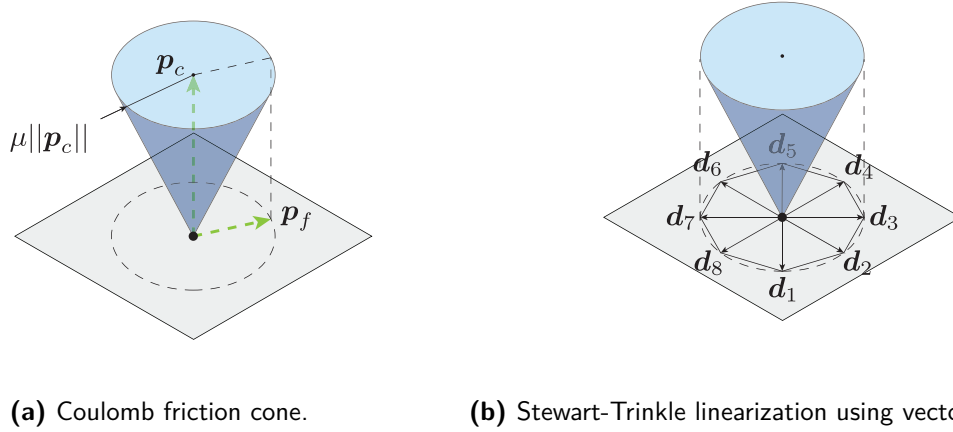
where  $f_c$  represents the contact normal force and  $\mu$  the Coulomb friction coefficient. The friction force can take on any direction within the contact plane. However, if the body is sliding, the magnitude is given by

$$f_f = \mu f_c, \quad (4-23)$$

and the direction is opposite the relative velocity.

The total contact force (contact normal force and contact friction force) is therefore bounded by a so called friction cone, as illustrated in Figure 4-2. The slope of the cone is equal to the friction coefficient  $\mu$ . When the contact is rolling, the total contact force can be anywhere in the friction cone. When the contact is sliding, the friction force is maximal, and therefore the contact force will be somewhere along the perimeter of the friction cone, and opposite to the relative sliding velocity.

<sup>1</sup> Although somewhat confusing, in literature a rolling contact often refers to a situation where at the contact point there is no relative velocity tangential to the contact plane. Think of a block lying still on a inclined floor. Contact points between the block and the floor are also considered to be 'rolling'.



**Figure 4-2:** The friction cone, with height  $\|p_c\|$  and radius  $\mu\|p_c\|$ , and the Stewart-Trinkle linearization.

#### 4-4-1 Linearized friction model

The friction cone is a quadratic shape and has to be linearized before we can include it in our linear system. In the Stewart-Trinkle method, this is done by approximating the cone by a set of vectors  $[d_1, \dots, d_\eta]$ , where  $\eta$  is the number of vectors used to approximate the friction cone. This is illustrated in figure Figure 4-2. We concatenate these vectors such that

$$D = [d_1, \dots, d_\eta]. \quad (4-24)$$

We define a vector  $\lambda_f \in \mathbb{R}^\eta$ , whose entries  $\lambda_{f,i}$  correspond to the impulse contribution of the friction in direction  $d_i$ . The total friction impulse  $p_f$  is therefore

$$p_f = D\lambda_f \quad (4-25)$$

Using this set of unit vectors, the Stewart-Trinkle method combines all the characteristics of Coulomb's friction model in the very elegant set of equations

$$D^T u^{n+1} + E\gamma = \sigma, \quad \sigma \geq 0, \quad (4-26a)$$

$$\mu\lambda_c - E^T \lambda_f = \zeta, \quad \zeta \geq 0, \quad (4-26b)$$

together with the following complementarity conditions

$$\lambda_f \geq 0, \quad \sigma^T \lambda_f = 0, \quad (4-26c)$$

$$\gamma \geq 0, \quad \zeta \gamma = 0. \quad (4-26d)$$

Here  $E \in \mathbb{R}^\eta$  represents a vector filled with ones. The symbols  $\sigma$  and  $\zeta$  are just implemented to simplify the complementarity notation, and have no clear physical meaning.

The physical meaning of  $\gamma$  depends on the situation. When sliding one can see it as an approximation of the relative velocity in the tangential plane. When rolling, it can take any value larger than or equal to zero, and has no physical meaning.

Eq. (4-26b) can be interpreted as the constraint that ensures that the magnitude of the friction force does not exceed what the relation with the between the normal impulse  $\lambda_c$  and the friction coefficient  $\mu$  dictates.<sup>2</sup>

The purpose of Eq. (4-26a) is to ensure that friction force and relative sliding velocity are opposite in direction.

We can now include the conditions for friction in Eq. (4-21).

$$\underbrace{\begin{bmatrix} \mathbf{M} & -\mathbf{J}_e^T & -\mathbf{J}_c^T & -\mathbf{D} & 0 \\ \mathbf{J}_e & 0 & 0 & 0 & 0 \\ \mathbf{J}_c & 0 & 0 & 0 & 0 \\ \mathbf{D}^T & 0 & 0 & 0 & \mathbf{E} \\ 0 & 0 & \mu & -\mathbf{E}^T & 0 \end{bmatrix}}_A \underbrace{\begin{bmatrix} \mathbf{u}^{n+1} \\ \lambda_e \\ \lambda_c \\ \lambda_f \\ \gamma \end{bmatrix}}_x - \underbrace{\begin{bmatrix} \mathbf{M}\mathbf{u}^n + h\mathbf{f}_{\text{ext}} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_b = \underbrace{\begin{bmatrix} 0 \\ 0 \\ a \\ \boldsymbol{\sigma} \\ \zeta \end{bmatrix}}_w \quad (4-27)$$

$$\begin{bmatrix} a \\ \boldsymbol{\sigma} \\ \zeta \end{bmatrix} \geq 0 \quad , \quad \begin{bmatrix} \lambda_c \\ \lambda_f \\ \gamma \end{bmatrix} \geq 0 \quad , \quad \begin{bmatrix} a \\ \boldsymbol{\sigma} \\ \zeta \end{bmatrix}^T \begin{bmatrix} \lambda_c \\ \lambda_f \\ \gamma \end{bmatrix} \geq 0$$

The system will give us joint, friction and contact forces over time interval  $h$ , and the velocity at the end of the time-step. This velocity can be plugged into the position update to give the position of the body at  $n + 1$ .<sup>3</sup>

The system is an approximation of Coulomb's friction model that improves with the number of vectors that span the friction cone ( $\eta$ ) because of two reasons. First, in the event of rolling the magnitude is underestimated if the friction direction is between two vectors  $\mathbf{d}_i$ . Second, in the event of sliding relative velocity is forced in the middle between the two vectors  $\mathbf{d}_i$  nearest to the actual direction according to Coulomb's model. This is shown in example 1.

### Example 1. Linearized friction.

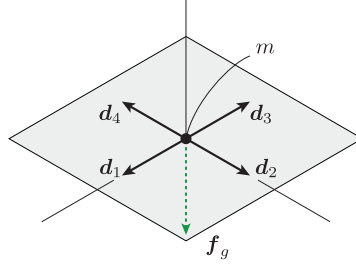
*Let us examine an example of the Stewart-Trinkle method with the linearized friction model. Consider a point-mass with mass  $m$  at rest on a plane with friction coefficient  $\mu$ , subject to a gravitational acceleration  $g$ .*

*Since we are discussing a point-mass, it will suffice to only describe translations. Our generalized velocity vector  $\mathbf{u} \in \mathbb{R}^3$  and generalized position vector  $\mathbf{s} \in \mathbb{R}^3$  only contain translations, and our generalized mass matrix can be simplified to*

$$\mathbf{M} = \begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & m \end{bmatrix}. \quad (4-28)$$

<sup>2</sup>Recall that  $\lambda_c$  only represents the magnitude of the normal impulse if  $\mathbf{J}_c^T$  is normalized, such that its magnitude is equal to one. It should be noted that therefore, normalizing these Jacobians is imperative to ensure physical correctness of the friction model.

<sup>3</sup>It should be noted that for readability, transformations from body d.o.f. to contact point velocities have been omitted. The reader is referred to the work of K. Erleben for a more complete notation [22].



**Figure 4-3:** Point mass lying on a plane. Note the four matrices  $d_i$  used for the linearized friction model, and the gravity force  $\mathbf{f}_{\text{ext}}$  acting on the point-mass.

The mass is lying on a horizontal plane, yielding a unilateral constraint  $\psi(\mathbf{s})$ , such that

$$\psi(\mathbf{s}) = s_3 \geq 0, \quad (4-29)$$

$$\mathbf{J}_c = \frac{\partial \psi(\mathbf{s})}{\partial \mathbf{s}} = [0, 0, 1]. \quad (4-30)$$

Our friction cone will be described by four unit vectors  $d_i$ , yielding

$$\mathbf{D} = [d_1, d_2, d_3, d_4] = \begin{bmatrix} 0 & 1 & 0 & -1 \\ 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \quad (4-31)$$

Recall that we have defined

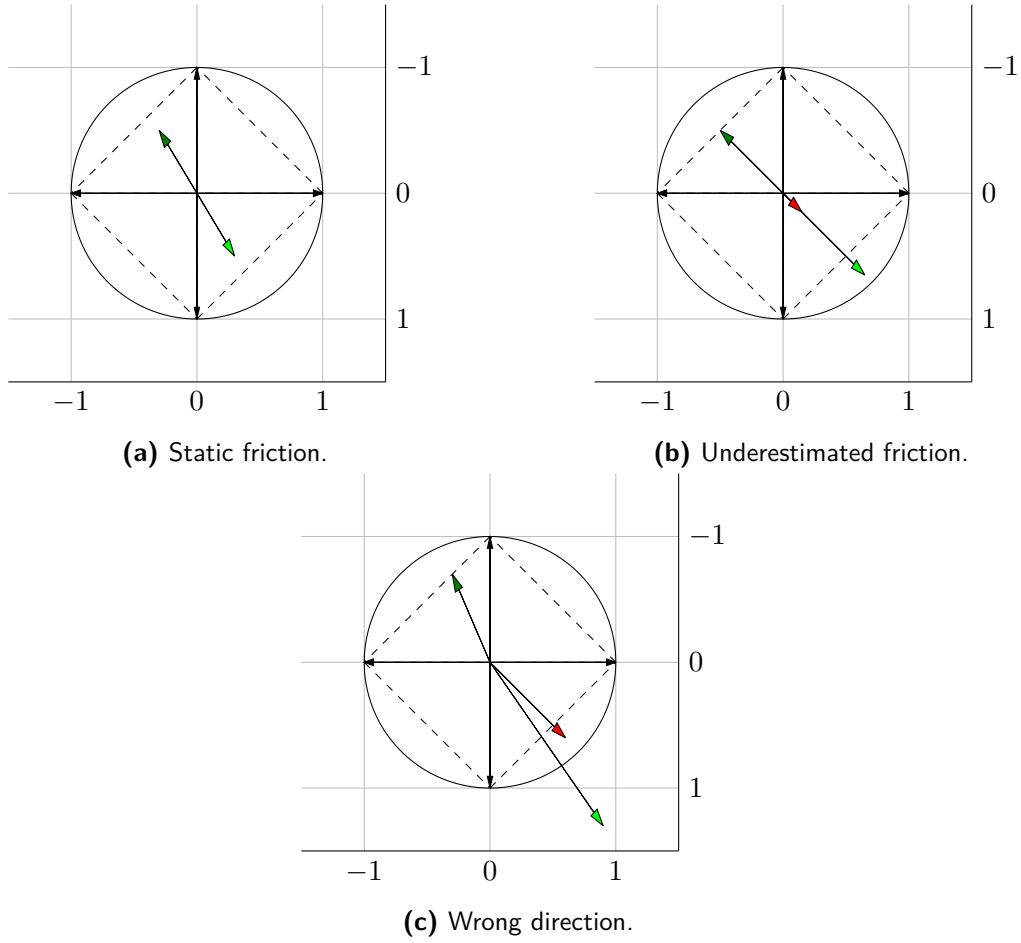
$$\mathbf{E} = [1, 1, 1, 1]^T. \quad (4-32)$$

Before the MLCP presented in Eq. (4-27) can be solved using Lemke's algorithm (see section 5-2), it needs to be rewritten to a Linear Complementarity Problem (LCP) by writing the equations of motion explicitly for the velocity at the next time-step ( $\mathbf{u}^{n+1}$ ) and plugging it into the constraint equations. Eliminating the terms corresponding to (not present) bilateral constraints and rewriting yields

$$\begin{bmatrix} \mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T & \mathbf{J}_c \mathbf{M}^{-1} \mathbf{D} & \mathbf{0} \\ \mathbf{D}^T \mathbf{M}^{-1} \mathbf{J}_c^T & \mathbf{D}^T \mathbf{M}^{-1} \mathbf{D} & \mathbf{E} \\ \mu & -\mathbf{E}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \lambda_c \\ \lambda_f \\ \gamma \end{bmatrix} - \begin{bmatrix} -\mathbf{J}_c(\mathbf{u}^n + h\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}) \\ -\mathbf{D}^T(\mathbf{u}^n + h\mathbf{M}^{-1}\mathbf{f}_{\text{ext}}) \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} a \\ \boldsymbol{\sigma} \\ \zeta \end{bmatrix}. \quad (4-33)$$

This problem can be solved using Lemke's algorithm. Depending on the magnitude of the horizontal external force we apply on the point-mass we can classify three different types of outcomes, as depicted in Figure 4-4. When the external force applied on the point-mass is within the square area depicted by the black dashed line, static friction is modeled accurately. When external force is beyond this border, but is still within the circle that limits friction according to Coulomb's friction model, we see that static friction is underestimated, and the point-mass starts to slide.

Furthermore, we see that the direction of dynamic friction is miss-judged in this method. The sliding velocity will position itself in the exact middle between two of the unit vectors  $d_i$ . As discussed, this error can be reduced by taking more vectors  $d_i$ .



**Figure 4-4:** Friction impulse  $\mathbf{p}_f$  (dark green), external applied impulse  $h\mathbf{f}_{\text{ext}}$  (light green) and velocity  $\mathbf{u}^{n+1}$  (red), compared to the actual boundary on Coulomb friction (black circle) and the unit vectors  $\mathbf{d}_i$  used for the linearization (black arrows).

#### 4-4-2 Decoupled friction model

The system presented in section 4-4-1 is shown to always have a solution for Lemke's algorithm [21, 36]. However, faster algorithms like Projected Gauss-Seidel (PGS) require the system to be symmetric, which is not the case due to the relation between the normal impulse and friction impulse through the friction coefficient ( $\mu$ ). Therefore, an even further reduced model to represent friction was developed.

In this model, we treat the friction as two independent one-dimensional friction models. We describe our friction cone by using two orthogonal vectors  $\mathbf{d}_1$  and  $\mathbf{d}_2$ , each with their corresponding friction impulse component  $\lambda_{f1}$  and  $\lambda_{f2}$ . If we concatenate the two orthogonal vectors into a single matrix  $\mathbf{D}$ , and the respective friction impulse components into a vector  $\boldsymbol{\lambda}_f = [\lambda_{f1}, \lambda_{f2}]$ , we can again write our total friction impulse as

$$\mathbf{p}_f = \mathbf{D}\boldsymbol{\lambda}_f, \quad (4-34)$$

and both directions are constrained by

$$-\mu\lambda_c \leq \lambda_{f1} \leq \mu\lambda_c \quad (4-35a)$$

$$-\mu\lambda_c \leq \lambda_{f2} \leq \mu\lambda_c \quad (4-35b)$$

This yields a new system

$$\underbrace{\begin{bmatrix} M & -J_e^T & -J_c^T & -D \\ J_e & 0 & 0 & 0 \\ J_c & 0 & 0 & 0 \\ D^T & 0 & 0 & 0 \end{bmatrix}}_A \underbrace{\begin{bmatrix} \mathbf{u}^{n+1} \\ \lambda_e \\ \lambda_c \\ \boldsymbol{\lambda}_f \end{bmatrix}}_x - \underbrace{\begin{bmatrix} M\mathbf{u}^n + h\mathbf{f}_{\text{ext}} \\ 0 \\ 0 \\ 0 \end{bmatrix}}_b = \underbrace{\begin{bmatrix} 0 \\ 0 \\ a \\ \boldsymbol{\sigma} \end{bmatrix}}_w \quad (4-36)$$

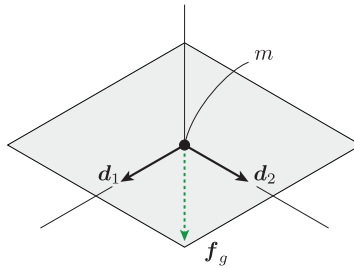
$$\begin{bmatrix} -\infty \\ -\infty \\ 0 \\ -\mu\lambda_c \end{bmatrix} \leq \begin{bmatrix} \mathbf{u}^{n+1} \\ \lambda_e \\ \lambda_c \\ \boldsymbol{\lambda}_f \end{bmatrix} \leq \begin{bmatrix} \infty \\ \infty \\ \infty \\ \mu\lambda_c \end{bmatrix}$$

This system is nearly a MLCP as defined in definition 1, with the exception of the relation between the boundary on the friction impulse and the normal impulse. Due to this relation, it is technically a Mixed Non-linear Complementarity Problem (MNCP). This problem can be handled by using the fact that the problem can be solved iteratively, first computing  $\lambda_c$ , and using that to define the boundary on  $\lambda_{f,i}$ .

The decoupled friction model is also an approximation of Coulomb's friction model. Contrary to the linearized friction model, when rolling, friction will actually be overestimated if the direction is not in line with either of the two the principal directions ( $\mathbf{d}_1$  or  $\mathbf{d}_2$ ). When sliding in a direction not in line with the principal directions, the friction will be forced into the corner of the friction pyramid. More on this simplified friction model can be found in the work done by Silcowitz [37].

### Example 2. Decoupled friction.

Now let us examine an implementation of the decoupled friction model. Same as in the previous example, we assume a point-mass lying on a flat plane, depicted in Figure 4-5. Our



**Figure 4-5:** Point mass lying on a plane. Note the four matrices  $d_i$  used for the decoupled friction model, and the gravity force  $\mathbf{f}_{\text{ext}}$  acting on the point-mass.

generalized position vector, velocity vector and mass matrix are identical. Furthermore the



Jacobian  $\mathbf{J}_c$  of the unilateral constraint is identical. The matrix  $\mathbf{D}$  is built up out of only two vectors, such that

$$\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_1] = \begin{bmatrix} 0 & 1 \\ 1 & 0 \\ 0 & 0 \end{bmatrix} \quad (4-37)$$

Again we rewrite the system explicitly for the unknown impulses, yielding

$$\underbrace{\begin{bmatrix} \mathbf{J}_c \mathbf{M}^{-1} \mathbf{J}_c^T & \mathbf{J}_c \mathbf{M}^{-1} \mathbf{D} \\ \mathbf{D}^T \mathbf{M}^{-1} \mathbf{J}_c^T & \mathbf{D}^T \mathbf{M}^{-1} \mathbf{D} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \lambda_c \\ \lambda_f \end{bmatrix}}_{\mathbf{x}} - \underbrace{\begin{bmatrix} -\mathbf{J}_c(\mathbf{u}^n + h\mathbf{M}^{-1}\mathbf{f}_{ext}) \\ -\mathbf{D}^T(\mathbf{u}^n + h\mathbf{M}^{-1}\mathbf{f}_{ext}) \end{bmatrix}}_{\mathbf{b}} = \underbrace{\begin{bmatrix} a \\ \boldsymbol{\sigma} \end{bmatrix}}_{\mathbf{w}} \quad (4-38)$$

$$\begin{bmatrix} 0 \\ -\mu\lambda_c \end{bmatrix} \leq \begin{bmatrix} \lambda_c \\ \lambda_f \end{bmatrix} \leq \begin{bmatrix} \infty \\ \mu\lambda_c \end{bmatrix}$$

Finally, we need to define the boundaries on the friction impulses, which are dependent on variable  $\lambda_c$ . As discussed before, in an actual gaming engine this could be computed based on the value of  $\lambda_c$  computed at the previous time-step. We could also use an iterative solver that in every iteration first computes  $\lambda_c$ , and recomputes the boundaries based on this value. However, for simplicity in this example we will define it analytically, such that

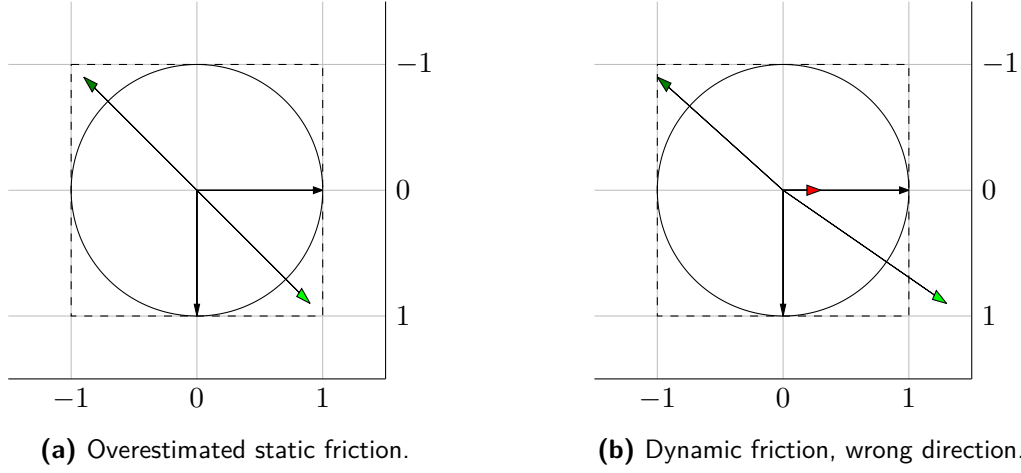
$$-h\mu mg \leq \lambda_c \leq h\mu mg \quad (4-39)$$

This system can be solved with Lemke's algorithm, but since it is positive symmetric definite, it can also be solved using the PGS algorithm, which will be explained later. In Figure 4-6a we see that in the case of static friction, the direction is modeled correctly. However, contrary to the linearized friction model, the maximum friction force magnitude is overestimated, since it is not limited by a circle, but in fact by a square with sides  $2\mu\lambda_c$ .

When we increase the force, we obtain dynamic friction, and the body starts to slide, as is shown in Figure 4-6b. Since it is a decoupled problem, we see that it treats both directions separately, resulting again in a wrong direction of the friction force and computed velocity. Contrary to the linearized system, this error cannot be reduced by taking more vectors  $\mathbf{d}_i$  to model the system.

## 4-5 Elastic collisions

The system described above can only handle so called inelastic collisions. This refers to collisions where, upon impact, the relative velocity between two bodies is reduced to zero. If no other forces act on the bodies, they would continue to travel together. A different type of



**Figure 4-6:** Friction impulse  $\mathbf{p}_f$  (dark green), external applied impulse  $h\mathbf{f}_{\text{ext}}$  (light green) and velocity  $\mathbf{u}^{n+1}$  (red), compared to the actual boundary on Coulomb friction (black circle) and the unit vectors  $\mathbf{d}_i$  used for the decoupling (black arrows).

collision is an elastic collision. Here some of the initial velocity is actually reversed. Think of a bouncing ball. One way to model this is through Newton's hypothesis [16].

Newton's hypothesis relates the relative velocity between two bodies normal to the contact plane before and after the collision through a so called coefficient of restitution  $\epsilon \in [0, 1]$ . It states

$$\mathbf{v}^+ = -\epsilon \mathbf{v}^-, \quad (4-40)$$

where  $\mathbf{v}^-$  and  $\mathbf{v}^+$  refer to respectively the velocity of the contact point directly before and directly after the collision. A value of zero corresponds to a fully inelastic collision (i.e. no bounce), and a value of one corresponds to a fully elastic collision (i.e. no energy is lost). We can redefine our unilateral constraint (Eq. (4-17)) as

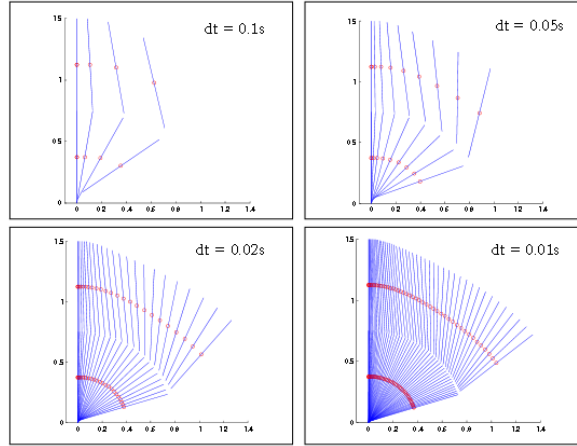
$$\mathbf{J}_c \mathbf{u}^{n+1} \geq \underbrace{-\epsilon \mathbf{J}_c \mathbf{u}^n}_{b_e}. \quad (4-41)$$

This relation can be used to augment either the MLCP (Eq. (4-27)) or the MNCP (Eq. (4-36)), by adding the right-hand side  $b_e$  to the third entry of vector  $\mathbf{b}$ . This constrains the relative normal velocity just after impact to satisfy Newton's hypothesis.

The variable  $a$  however loses its value physical meaning as relative velocity of the contact point at  $n+1$ , but instead represents the difference between the velocity dictated by Eq. (4-40) and the simulated velocity at  $n+1$ .

## 4-6 Constraint stabilization

Since this method uses maximal coordinate formulation and integrates our motion numerically, both the equality as well as the inequality constraints will experience a numerical error known as drift. For joints this manifests itself as bodies slowly moving apart, illustrated in figure 4-7. For non-penetration constraints it will manifest itself as bodies penetrating.



**Figure 4-7:** An example of joint drift [38].

Drift will be reduced by taking a smaller time-step size, but will never be fully eliminated. This can be overcome by applying a form of constraint stabilization. The two most used methods are *Baumgarte stabilization* and *Post-stabilization*.

#### 4-6-1 Baumgarte stabilization

Remember our joint constraint equation (4-10). Here we look at the constraint on a velocity level. Baumgarte stabilization on a velocity level takes the form

$$\mathbf{J}_e \mathbf{u}^{n+1} + \alpha \phi(\mathbf{s}^n) = 0, \quad (4-42)$$

where  $\alpha$  is a stabilization parameter that can be set to obtain different levels of stabilization [26].<sup>4</sup> Note that if we choose a value of one for  $\alpha$ , this notation is equal to the first order Taylor expansion without setting the first term to zero (Eq. (4-8)). Crucial for this method to behave correctly is finding the right value for  $\alpha$ , but on closer inspection we can reveal an interesting relation between  $\alpha$  and the time-step size  $h$  [22]. If we choose

$$\alpha = \frac{k_{\text{erp}}}{h} \quad (4-43)$$

we obtain

$$\mathbf{J}_e \mathbf{u}^{n+1} = \underbrace{-\frac{k_{\text{erp}}}{h} \phi(\mathbf{s}^n)}_{b_B}. \quad (4-44)$$

Here  $k_{\text{erp}}$  is referred to as the error reduction parameter. If we set this to a value of one, we see that the error reduction term will eliminate the existing error in one time-step. However, this is only true if the Jacobian is linear with respect to  $\mathbf{s}$ , which is not always the case.

<sup>4</sup>Another very common way of writing the Baumgarte stabilization method is on an acceleration level. In this case we would write  $\ddot{\phi} + \alpha \dot{\phi} + \beta \phi = 0$ , where  $\phi$  represents some arbitrary constraint, and  $\alpha$  and  $\beta$  are two separate stabilization parameters.

Therefore choosing the right parameter is still crucial, but the behavior has been decoupled from the time-step size.

This method can be applied in the same manner to complementarity problem as with we did with elastic collisions, by adding the term  $b_B$  to the third entry of vector  $\mathbf{b}$  of the complementarity problem.

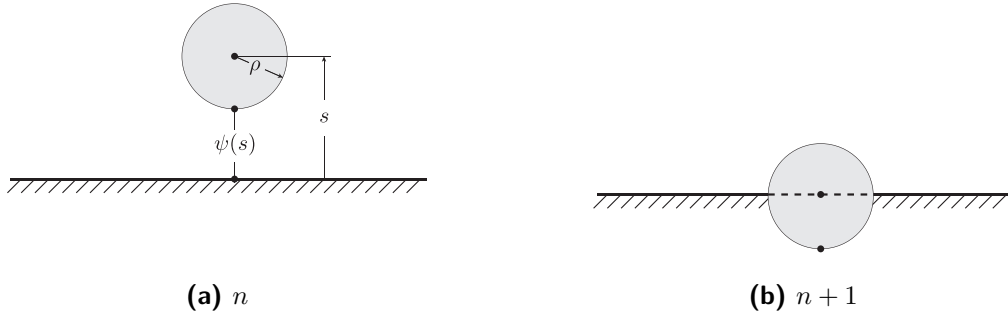
Baumgarte error reduction can also be applied to inequality constraints, in the form

$$\mathbf{J}_e \mathbf{u}^{n+1} \geq \underbrace{-\frac{k_{\text{erp}}}{h} \psi(\mathbf{s}^n)}_{b_B}, \quad (4-45)$$

although we will later see that this can lead to un-physical behavior.

**Example 3. Baumgarte stabilization of a unilateral constraint.**

Let us discuss an example of Baumgarte stabilization applied to a unilateral constraint. We consider a simple spherical body with one degree of freedom with  $m = 1$  under the influence of gravitational acceleration  $g = -9.81 \text{ [m/s}^2\text{]}$  hitting a plane, as illustrated in Figure 4-8a. Since our system is one dimensional, the state can be described by a scalar  $s$  for position of



**Figure 4-8:** Illustration of sphere falling towards a plane and penetrating.

the Centre Of Mass (COM) and  $u$  for the velocity. Our unilateral constraint is written as

$$\psi(s) = s - \rho \geq 0, \quad (4-46)$$

$$\mathbf{J}_c = 1, \quad (4-47)$$

where  $\rho = 1$  is defined as the radius of the sphere.

We define initial conditions at time-step  $n$  as

$$s^n = 9.81 \quad (4-48)$$

$$u^n = 0 \quad (4-49)$$

At time-step  $n$  our bodies are still separated, no collisions are detected, and the contact is inactive. Integrating to  $n + 1$  with time-step size  $h = 1$  yields the following condition

$$u^{n+1} = -9.81, \quad (4-50)$$

$$s^{n+1} = 0, \quad (4-51)$$

$$\psi(s^{n+1}) = -1. \quad (4-52)$$

As can be seen in Figure 4-8b, the bodies are penetrating, and the collision detection sets the contact to active. We now need to build up our MNCP for the problem. Combined with the Baumgarte stabilization term this yields

$$\begin{bmatrix} \mathbf{M} & -\mathbf{J}_c^T \\ \mathbf{J}_c & 0 \end{bmatrix} \begin{bmatrix} u^{n+2} \\ \lambda_c \end{bmatrix} - \begin{bmatrix} \mathbf{M}u^{n+1} + hmg \\ -\frac{k_{erp}}{h}\psi(s^{n+1}) \end{bmatrix} = \begin{bmatrix} 0 \\ a \end{bmatrix} \quad (4-53)$$

$$a \geq 0 \quad , \quad \lambda_c \geq 0 \quad , \quad a^T \lambda_c = 0$$

Since the value of our constraint  $\psi(s^{n+1})$  is smaller than zero, we need at least a larger than zero velocity  $u^{n+2}$  such that

$$\mathbf{J}_c u^{n+2} \geq -\frac{k_{erp}}{h}\psi(s^{n+1}), \quad (4-54)$$

Since we need positive, non-zero velocity at  $n+2$  in order for the system to have a correct solution, and we have a negative velocity at  $n+1$ , we need a positive, non-zero Lagrange multiplier  $\lambda_c$ . Through the complementarity this yields that

$$a = 0, \quad (4-55)$$

and therefore

$$\mathbf{J}_c u^{n+2} = -\frac{k_{erp}}{h}\psi(s^{n+1}), \quad (4-56)$$

$$u^{n+2} = -\mathbf{J}_c^{-1} \frac{k_{erp}}{h}\psi(s^{n+1}). \quad (4-57)$$

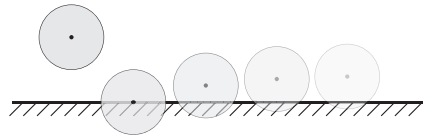
Setting  $k_{erp} = 0.2$ , plugging the values and integrating the motion leads to the new state

$$u^{n+2} = 0.2, \quad (4-58)$$

$$s^{n+2} = 0.2, \quad (4-59)$$

$$\psi(s^{n+2}) = -0.8. \quad (4-60)$$

If we repeat this process over several time-steps we will see that the penetration is gradually and asymptotically reduced by the constraint stabilization term. This is illustrated in Figure 4-9.



**Figure 4-9:** Stabilization behavior due to Baumgarte stabilization.

### 4-6-2 Post-stabilization

Another method is to follow each motion integration step with a separate stabilization step. This method was first proposed by M. Cline and D. Pai [26]. The main difference is that instead of altering constraints on the *velocities*, the stabilization step takes the un-stabilized output of the motion integration as input, and alters the new *position* to satisfies the constraints.

In our stabilization step we wish to find some state update  $\Delta \mathbf{s}$  such that  $\phi(\mathbf{s} + \Delta \mathbf{s}) = 0$ . We can linearize our constraint such that

$$\phi(\mathbf{s} + \Delta \mathbf{s}) \approx \phi(\mathbf{s}) + \frac{\partial \phi(\mathbf{s})}{\partial \mathbf{s}} \Delta \mathbf{s} = 0, \quad (4-61)$$

which represent the first order Taylor expansion of the constraint.<sup>5</sup> We define

$$\mathbf{G}(\mathbf{s}) = \frac{\partial \phi(\mathbf{s})}{\partial \mathbf{s}}. \quad (4-62)$$

Rearranging we see that our stabilization step  $\Delta \mathbf{s}$  should satisfy

$$\mathbf{G}(\mathbf{s}) \Delta \mathbf{s} = -\phi(\mathbf{s}). \quad (4-63)$$

which is not easy to solve, due to the fact that  $\mathbf{G}$  is usually not symmetric. Expensive matrix methods for solving this system can be applied, but a better way is to formulate it as a linear system, that will be solved separately from the main MNCP [26]. We write

$$-\begin{bmatrix} \mathbf{1} & -\mathbf{G}(\mathbf{s})^T \\ \mathbf{G}(\mathbf{s}) & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{s} \\ \lambda \end{bmatrix} = \begin{bmatrix} 0 \\ \phi(\mathbf{s}) \end{bmatrix}. \quad (4-64)$$

This method can be easily developed to include unilateral constraints as well, in which case it will be become a LCP.

### 4-6-3 Stabilization method comparison

There are a few important trade-offs between the two methods [26]:

- Baumgarte stabilization alters velocities, whereas post-stabilization alters position. Therefore Baumgarte stabilization can have an effect on the kinetic energy of the system, and in the event of collisions, can actually increase energy. Post-stabilization effects potential energy, in the event that a forcefield acts on the body.
- Baumgarte stabilization uses the special constant  $k_{\text{erp}}$ , that needs to be carefully selected. Post-stabilization does not have parameter that needs to be set.
- With post-stabilization, the error is usually eliminated in the same step as it is created. In Baumgarte stabilization, the constraint error cannot effect the system until one step later. The time it takes for an error to be eliminated varies with the special constants, and is usually spread out over several time-steps. Also steady-state errors can occur.

---

<sup>5</sup>Note that  $\frac{\partial \phi(\mathbf{s})}{\partial \mathbf{s}} \neq \mathbf{J}$ , since  $\mathbf{J}$  also includes a transformation between twists and the orientation description using the quaternion.

- Baumgarte stabilization is easier to implement. Since it only requires evaluation of the constraint violation  $\phi(\mathbf{s}^n)$ , it does not have a large impact on computational expenses. Post-stabilization on the other hand requires solving another system of equations, which is more computationally expensive.
- Post-stabilization can perform poorly if constraints errors become too large. This is due to the fact that the linearization in Eq. (4-61) is not a good approximation. This can be overcome by limiting the size of the state update  $\Delta\mathbf{s}$ , and perform several steps per time-step, but of course this results in increased computational expense.





# The complementarity problem

Since constraint based simulation is built around a complementarity problem, it is important to understand these types of problems. This chapter aims to give a basic explanation of what a complementarity problem is and how these are typically solved in gaming engines.

## 5-1 Linear and mixed-linear complementarity problems

The complementarity problem knows several forms. The most basic form is the Linear Complementarity Problem (LCP), formulated in definition 1.

**Definition 1. Linear complementarity problem.** *Given a symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , a vector  $\mathbf{b} \in \mathbb{R}^n$ . Find  $\mathbf{x} \in \mathbb{R}^n$  and  $\mathbf{w} \in \mathbb{R}^n$  such that*

$$\mathbf{w} = \mathbf{Ax} - \mathbf{b}, \quad (5-1a)$$

$$\mathbf{w}^T \mathbf{x} = 0, \mathbf{w} \geq 0, \mathbf{x} \geq 0. \quad (5-1b)$$

The problem defined in Eq. (4-36) is said to be a Mixed Non-linear Complementarity Problem (MNCP). However, through a simple trick it can be treated as a Mixed Linear Complementarity Problem (MLCP) (see section 5-3). A versatile definition for a MLCP is given in definition 2 [22].

**Definition 2. Mixed linear complementarity problem.** *Given a symmetric matrix  $\mathbf{A} \in \mathbb{R}^{n \times n}$ , a vector  $\mathbf{b} \in \mathbb{R}^n$ , a vector of lower limits  $\mathbf{x}_l \leq 0$  and a vector of upper limits  $\mathbf{x}_u \geq 0$ , where  $\mathbf{x}_l, \mathbf{x}_u \in \mathbb{R}^n$ . Find  $\mathbf{x} \in \mathbb{R}^n$  and  $\mathbf{w} \in \mathbb{R}^n$  such that*

$$\mathbf{w} = \mathbf{Ax} - \mathbf{b}, \quad (5-2a)$$

$$\mathbf{x}_l \leq \mathbf{x} \leq \mathbf{x}_u, \quad (5-2b)$$

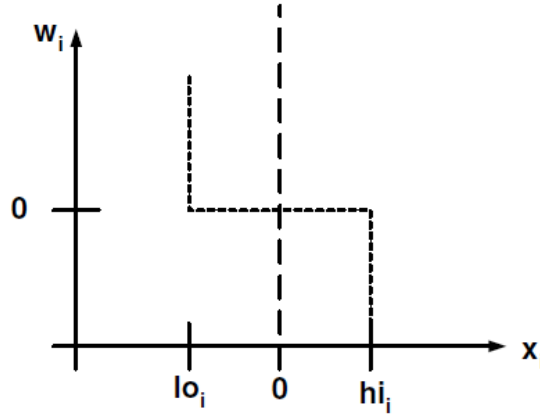
and for all  $i = 1, 2, \dots, n$ , one of the three conditions below holds

$$x_i = x_{l_i} \rightarrow w_i \geq 0, \quad (5-2c)$$

$$x_i = x_{h_i} \rightarrow w_i \leq 0, \quad (5-2d)$$

$$x_{l_i} < x_i < x_{h_i} \rightarrow w_i = 0. \quad (5-2e)$$

The complementarity conditions are clarified in figure 5-1. If  $x_i$  is within the boundaries then  $w_i = 0$ . If however  $x_i$  is on one of the boundaries,  $w_i$  is either unbounded below or above, depending on which boundary  $x_i$  is.



**Figure 5-1:** Illustration of the complementarity condition on  $x_i$  and  $w_i$  [22].

The difference between a LCP and a MLCP is that the latter can contain equalities, as well as inequalities. Usually a MLCP can be rewritten to form a LCP.

## 5-2 Linear complementarity problem solvers

One can divide all known linear complementarity problem solvers into three subclasses [16]:

- **Direct methods.** Direct methods can provide very accurate, or even exact solutions, but at a very high computational expense. Usually convergence time is in the order of  $\mathcal{O}(K^3)$ , where  $K$  is the total number of contact points. Examples of direct methods are Lemke's algorithm and the Keller method [36, 35, 19].
- **Iterative fixed point schemes.** These methods approximate a solution to LCP iteratively, until certain convergence criteria are met, or a maximum number of iterations is reached. Convergence time for these methods is  $\mathcal{O}(K)$ . However, accuracy of the solutions is affected. Also, these solvers have more stringent convergence criteria. A well known and often used iterative solver is the *Project Gauss-Seidel (PGS) method*.
- **Newton methods.** Newton methods for linear complementarity problems also solve the system iteratively, achieving higher accuracy than Projected Gauss-Seidel (PGS), but at a slightly higher convergence time  $\mathcal{O}(K^2)$ . A widely used solver is the PATH solver [39].

Because PGS is the most used solver in gaming industry packages, we will only discuss this solver in detail. For more information on the other two types of solvers, the reader is referred to the work by J. Bender [16]. PGS has of course had many contributors, but the notations and algorithm used in this section are slight adaptations of the work by K. Erleben [22].

Before looking at solving linear complementarity problems, we will first look at iterative solvers in general. These have been around for a long time, and are historically known for handling sparse systems well. They are able to achieve coarse solutions for systems of equations with relatively low cost, but are slow at achieving high precision solutions [30]. An iterative solver can be described as a solver that iteratively improves some possible solution until either an acceptable solution is met, or the maximum number of iterations is reached. Mathematically we can state this as

$$\mathbf{x}^{k+1} = f(\mathbf{x}^k) \quad (5-3)$$

where superscript  $k$  denotes the iteration. Any solution  $\mathbf{x}^*$  to the iterative method satisfies

$$\mathbf{x}^* = f(\mathbf{x}^*). \quad (5-4)$$

Now consider some arbitrary linear system of equations

$$\mathbf{A}\mathbf{x} = \mathbf{b} \quad (5-5)$$

By using matrix-splitting we can write this set of linear equations in the form of equation 5-4, and use an iterative solver. First we split matrix  $\mathbf{A}$  a strictly lower diagonal matrix  $\mathbf{L}$ , a diagonal matrix  $\mathbf{D}$  and a strictly upper diagonal matrix  $\mathbf{U}$ , such that

$$\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}. \quad (5-6)$$

We can now rewrite our system as

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \quad (5-7a)$$

$$(\mathbf{L} + \mathbf{D} + \mathbf{U})\mathbf{x} = \mathbf{b}, \quad (5-7b)$$

$$\mathbf{D}\mathbf{x} = \mathbf{b} - (\mathbf{L} + \mathbf{U})\mathbf{x}, \quad (5-7c)$$

$$\mathbf{x} = \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x} \quad (5-7d)$$

which can be written in the form of 5-4

$$\mathbf{x}^{k+1} = f(\mathbf{x}^k), \quad (5-8)$$

and

$$f(\mathbf{x}^k) = \mathbf{D}^{-1}\mathbf{b} - \mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})\mathbf{x}^k, \quad (5-9)$$

In fact, we can update the system per individual row. The update of row  $i$  can be written as

$$x_i^{k+1} = \frac{\left(b_i - \sum_{j=1}^{i-1} L_{ij}x_j^k - \sum_{j=i+1}^n U_{ij}x_j^k\right)}{A_{ii}}. \quad (5-10)$$

This method is referred to as the Jacobi method. Upon closer inspection, we see that when we compute  $x_i^{k+1}$  for all  $j < i$ . Since these values are already known, the Jacobi algorithm can be improved by using the most recent values available, such that

$$x_i^{k+1} = \frac{\left(b_i - \sum_{j=1}^{i-1} L_{ij}x_j^{k+1} - \sum_{j=i+1}^n U_{ij}x_j^k\right)}{A_{ii}}. \quad (5-11)$$

This method is called the Gauss-Seidel method. The main difference between these two methods is that the Jacobi method is parallel in nature, whereas Gauss-Seidel is sequential. This method can also be written in matrix form

$$\mathbf{x}^{k+1} = \mathbf{D}^{-1} (\mathbf{b} - \mathbf{L}\mathbf{x}^{k+1} - \mathbf{U}\mathbf{x}^k). \quad (5-12)$$

We now have a method for solving normal linear systems, but it is not yet suited for our application. In order to solve MLCP's, we define a clamping operation.

**Definition 3. Clamping.** *Given a vector  $\mathbf{x} \in \mathbb{R}^n$ , a vector of lower limits  $\mathbf{x}_l \leq 0$  and a vector of upper limits  $\mathbf{x}_h \geq 0$ , where  $\mathbf{x}_l, \mathbf{x}_h \in \mathbb{R}^n$ . The clamping operation on  $\mathbf{x}$  is written  $(\mathbf{x})^+$ , and means for all  $i = 1, 2, \dots, n$*

$$x_i^+ = \max(\min(x_i, x_{u_i}), x_{l_i}) \quad (5-13)$$

Effectively this ensures that the value  $x_i$  is either between the upper and lower boundaries, or, in the case that it exceeds one of the boundaries, it is projected back to the closest boundary.

We can combine clamping and our previously determined Gauss-Seidel method, to solve a system described in definition 1. Lets define the element wise update of the normal Gauss-Seidel method as  $z_i^k$ , such that

$$z_i^k = \frac{(b_i - \sum_{j=1}^{i-1} L_{ij}x_j^{k+1} - \sum_{j=i+1}^n U_{ij}x_j^k)}{A_{ii}}. \quad (5-14)$$

Then the element wise update of a PGS solver is

$$x_i^{k+1} = \max(\min(z_i^k, x_{u_i}), x_{l_i}). \quad (5-15)$$

The method effectively *projects* the solution onto its boundaries, hence the name of the method.

A possible pseudocode implementation of this method is shown in algorithm 1. Here the iteration is looped until a fixed maximum number of iterations is met. One could also loop until certain convergence criteria are met. For more information on convergence criteria, and other types of iterative fixed point schemes, the reader is referred to the work by K. Erleben [23].

### 5-3 Treating a non-linear problem as a linear problem

As described in section 4-4-2, Eq. (4-36) is technically a non-linear problem, due to the relation between the normal force  $\lambda_c$  and the boundaries on the friction force. Remember

$$-\mu\lambda_c \leq \lambda_{f1} \leq \mu\lambda_c, \quad (5-16a)$$

$$-\mu\lambda_c \leq \lambda_{f2} \leq \mu\lambda_c. \quad (5-16b)$$

This section describes two ways to overcome this problem and treat the MNCP as if it were a MLCP.

**Algorithm 1** Projected Gauss-Seidel method

---

```

1: procedure PGS( $\mathbf{A}, \mathbf{b}, \mathbf{x}_l, \mathbf{x}_u, k_{\max}$ )
2:   Set  $x$  to initial guess
3:   for  $k = 1$  to  $k_{\max}$  do
4:     for  $i = 1$  to  $n$  do
5:        $z_i = 0$ 
6:       for  $j = 1$  to  $i - 1$  do ▷ Equivalent to  $\sum_{j=1}^{i-1} L_{ij}x_j^{k+1}$ 
7:          $z_i = z_i + A_{ij}x_j$ 
8:       end for
9:       for  $j = i + 1$  to  $n$  do ▷ Equivalent to  $\sum_{j=i+1}^n U_{ij}x_j^k$ 
10:         $z_i = z_i + A_{ij}x_j$ 
11:      end for
12:       $z_i = (b_i - z_i)/A_{ii}$ 
13:       $x_i = z_i$ 
14:      if  $x_i > x_{u_i}$  then ▷ Clamp operation of  $x_i$ 
15:         $x_i = x_{u_i}$ 
16:      end if
17:      if  $x_i < x_{l_{u_i}}$  then
18:         $x_i = x_{l_i}$ 
19:      end if
20:    end for
21:  end for
22: end procedure

```

---

**5-3-1 Decoupling the normal impulse and friction boundaries.**

The boundaries on the friction force in the contact problem at  $t = n$  can be defined based on the  $\lambda_c$  from the solution to the contact problem at  $t = n - 1$ . This reduces  $\lambda_c$  to a constant value that is set at the first iteration of the PGS algorithm, and decouples the friction boundaries from the normal impulse. The system is now transformed to a MLCP as defined in definition 1. However, this reduces the physical correctness of the problem, since the  $\lambda_c$  at  $t = n - 1$  is not necessarily the same as  $\lambda_c$  at  $t = n$ , and this does not converge when decreasing the size of the time-step  $h$ .

**5-3-2 Exploiting the iterative nature of the PGS algorithm.**

In the PGS algorithm, the rows of the complementarity problem are solved in a fixed order during each iteration. If in iteration  $k = 1$  we solve for the contact impulses first, we can use this approximation to define the boundaries on the friction impulses in that same iteration. Once we have updated all rows, and move to iteration  $k = 2$ , we again compute the normal impulses first and use these to redefine the boundaries on the friction force, and so on.

Although ones intuition could suggest that this algorithm would converge, no literature was found proving this. One could argue we are not solving a truly linear system in this manner. However, this seems to be a method that is used successfully in at least one gaming engine.<sup>1</sup>

---

<sup>1</sup>This is confirmed through email contact with Erwin Coumans, the creator of Bullet Physics. Also it is

## 5-4 Solution existence and uniqueness

The initial formulation by Stewart and Trinkle of the contact problem was shown to have a solution in most cases [20]. However, it wasn't until M. Anitescu and F. Potra used the Taylor approximation to describe the constraints on a velocity level that solution existence was guaranteed in all situations [21]. They proved that Lemke's algorithm was guaranteed to always find a solution to the problem.

In order for the PGS algorithm to have solution of the MLCP as defined in definition 1, there are two additional requirements.

- The matrix  $\mathbf{A}$  is symmetric.
- The matrix  $\mathbf{A}$  is positive definite.

After rewriting Eq. (4-36) explicitly for only the Lagrange multipliers, we obtain a system that fulfills these additional requirements.

Uniqueness of neither Eq. (4-27) nor Eq. (4-36) is not guaranteed [21]. This is however inherent in the rigid-body assumption. A physical way of interpreting this is by imagining a symmetric table with 4 identical legs standing on a plane. Due to the fact that we considered rigid-bodies only, the system is statically undetermined. If the legs were truly identical, also in their non-rigid behavior, the reaction force on each leg would be identical, and one-fourth of the total gravity force acting on the table. However, another solution to the problem we have formulated would be to have two legs of the table taking half of the total force each, and having the other two legs unloaded.

---

shown through computer experiments in chapter 6.

---

## Chapter 6

---

# Bullet & Blender

In chapters 4 and 5 the most important mathematical models involved in rigid-body simulation have been explained. Based on this knowledge we can identify which methods are currently used in Bullet & Blender, and what inaccuracies could arise from these models.

The first part of this chapter will identify what models are used by Bullet. The second part describes the loops through which Bullet & Blender cooperate. Finally, we will discuss what possible unphysical behavior can be present in simulations performed with Bullet & Blender.

1

### 6-1 Model identification

The aim of this chapter is to show which of the methods presented in chapter 4 are used by Bullet & Blender. Their inner workings have mainly been identified through comparison of outputs from simulations with Bullet & Blender to results obtained from implementations of the models in self-made Matlab scripts. Furthermore, some identification of these methods is based on correspondence with the creator of Bullet.

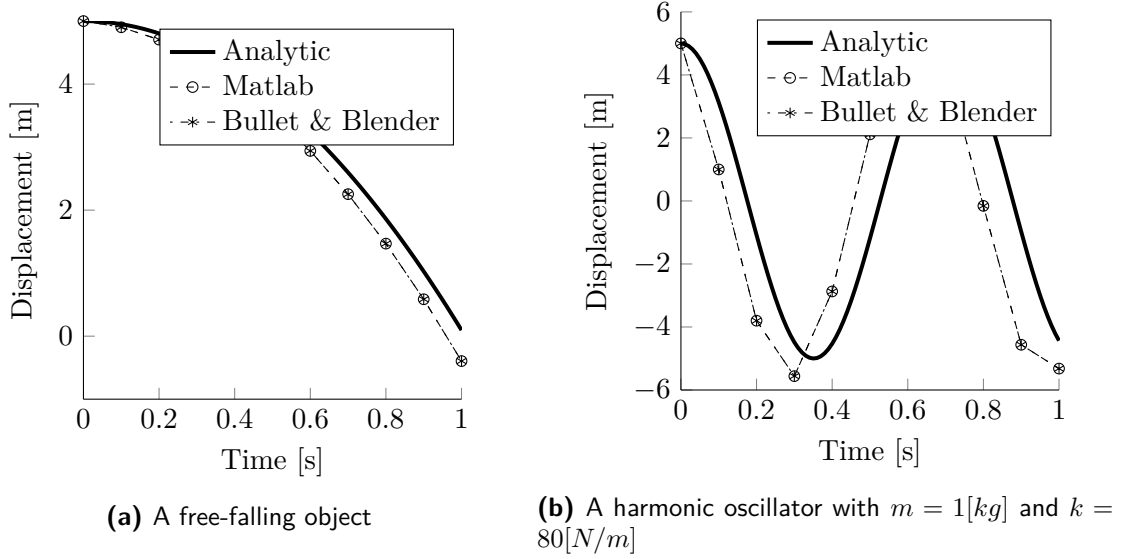
#### 6-1-1 Time-integration

Once all contact impulses have been obtained, Bullet integrates the motion of bodies through the semi-implicit Euler integration scheme. The best way to illustrate this is by taking two simple models, i.e. a falling sphere and a harmonic oscillator, and compare Bullet output to a Matlab implementation of the semi-implicit Euler method.

In Figure 6-1 it can be seen that Bullet & Blender indeed integrates the motion of these objects in the same way as a semi-implicit Euler scheme does. This is confirmed by literature that states that Bullet indeed uses this integration scheme[16].

---

<sup>1</sup>Note that this research is based on the version of Bullet integrated in Blender v2.67. This is not the most recent version, and possibly some of the inadequacies have already been addressed in newer versions of Bullet.

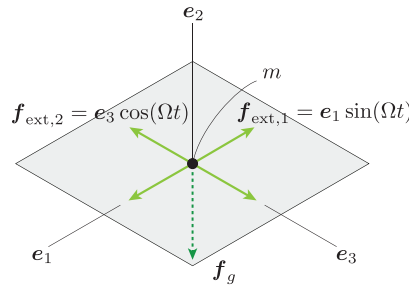


**Figure 6-1:** Displacements of two very simple simulations over time, comparing Bullet & Blender, a Matlab implementation of the semi-implicit Euler scheme and the analytic solution. In both simulations  $h = 0.1[\text{s}]$ .

### 6-1-2 Friction model

At the heart of Bullet & Blender we find the Stewart-Trinkle method, with the constraints on a velocity level, and a decoupled friction model, as defined in Eq. (4-36). To improve the accuracy of the friction model, the vectors in  $\mathbf{D}$  are aligned with the velocity at the beginning of every time-step. One could say that  $\mathbf{D}$  is defined in a *local* frame of reference.

We can illustrate this by simulating a point-mass sliding over a plane, with oscillating horizontal forces  $\mathbf{f}_{\text{ext},1}$  and  $\mathbf{f}_{\text{ext},2}$  acting on it, as illustrated in Figure 6-2. If we give the mass

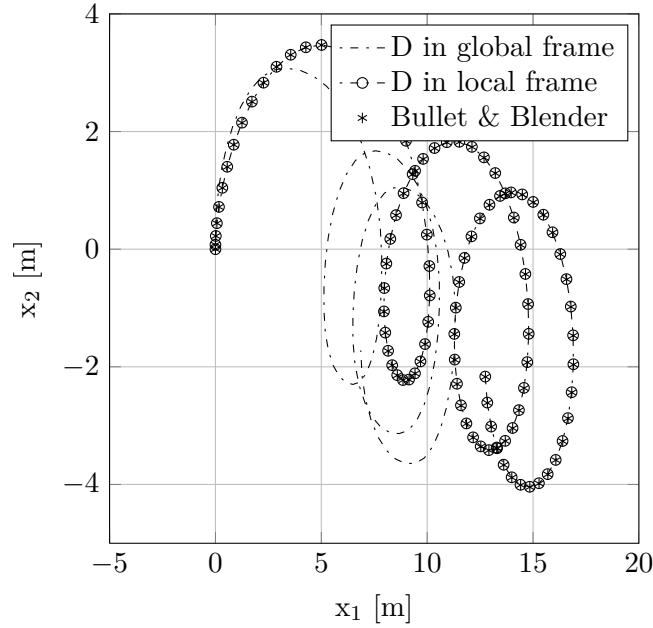


**Figure 6-2:** Illustration of friction experiment where a point-mass is moved across a plane under the influence of horizontal forces oscillating in time with excitation frequency  $\Omega$  (light green), and gravity (dark green). The contact point experiences friction with friction coefficient  $\mu$ .

a non-zero initial velocity at  $n = 0$ , it will result in elliptical trajectory across the plane. However, depending on the orientation of the principal directions of the friction model the exact trajectories will vary.

In Figure 6-3 it can be seen that friction in Bullet & Blender behaves in the same way as





**Figure 6-3:** Trajectory of an object moving over a plane with a oscillating external load in horizontal direction. Bullet & Blender compared to a Matlab implementation of Eq. (4-36) with a fixed or global  $\mathbf{D}$  and a local  $\mathbf{D}$ , aligned with the velocity.  $m = 1[kg]$ ,  $\mu = .25[-]$  and  $dt = 0.1[s]$

a Matlab implementation of the decoupled friction model, where the principal directions are aligned with the relative velocity.

### 6-1-3 Constraint stabilization

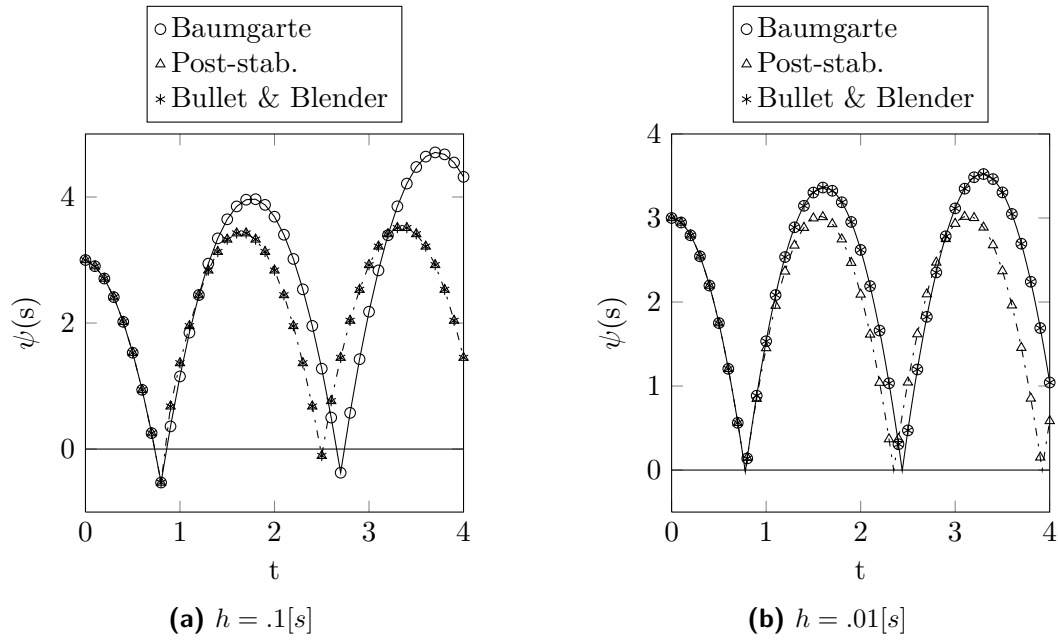
Bullet allows for both Baumgarte and post-stabilization to be used. In the the case of Baumgarte stabilization, Eq. (4-36) is augmented with an implementation the method as described in Eq. (4-44). Post-stabilization is applied, but with an additional parameter  $k_{erp} \in [0, 1]$  to reduce the stabilizing effect, probably to prevent instabilities from occurring.

Again, we illustrate what methods are applied in Blender & Bullet by looking at a simple example of an fully elastic object (i.e.  $\epsilon = 1$ ) bouncing on a static plane. We define a non-penetration constraint  $\psi(s)$  such that

$$\psi(s) = s - \rho \geq 0 \quad (6-1)$$

where  $s$  merely represent the height of the sphere Centre Of Mass (COM), and  $\rho$  the radius of the sphere. The results of this model simulated using Blender, and equivalent simulations with Baumgarte and post-stabilization are plotted in Figure 6-4.

One can detect that Blender seems to utilize both methods, most probably depending on the penetration depth at the contact. For large penetrations post-stabilization is used, for small penetrations Baumgarte is used.



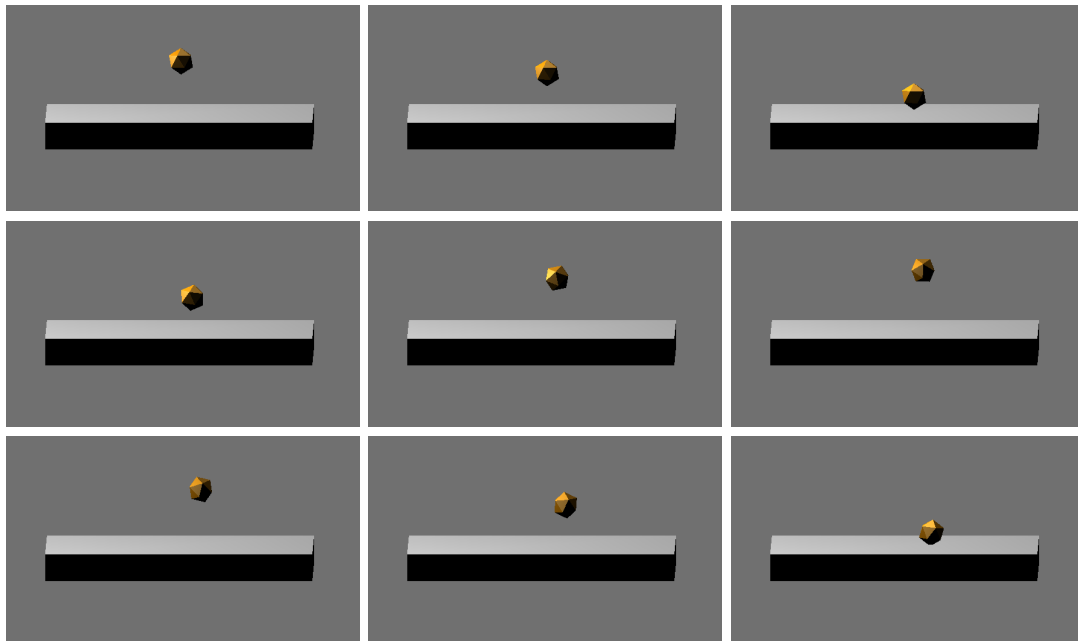
**Figure 6-4:** Value of the non-penetration constraint  $\psi(s)$  of a bouncing sphere over time. A value of the constraint of less than 0 corresponds to penetration. It can be seen that for a large time-step size and large penetration, Bullet & Blender behaves according to the post-stabilization method. However, for small time-step size and small penetration they behave according to the Baumgarte stabilization method. For post-stabilization  $k_{erp} = 0.8$  and for Baumgarte  $k_{erp} = 0.2$ . To clarify the figures the markers are shown every  $0.1[s]$ , irrespective of the simulation time-step size.

### 6-1-4 Complementarity problem solver

It is known from literature and direct contact with the author of Bullet that the version used in Blender uses an adaptation of the Projected Gauss-Seidel (PGS) algorithm as shown in algorithm 1. Due to the fact that Bullet is optimized for real-time animations, it uses a fixed number of iterations (i.e. 10) rather than some convergence criterion to end the iterative process.

Furthermore it exploits the iterative nature of the PGS algorithm to overcome the non-linearity in the Mixed Non-linear Complementarity Problem (MNCP) as described in section 5-3, solving the normal impulses first, and using them to redefine the boundaries on the friction in every iteration of the solver. This can be illustrated by simulating a collision with a sphere and a static plane. This time however, the sphere will have a initial horizontal velocity. Figure 6-5 shows screen shots from a simulation performed with Blender & Bullet, and Figure 6-6 the corresponding constraint value and angular velocity are shown.

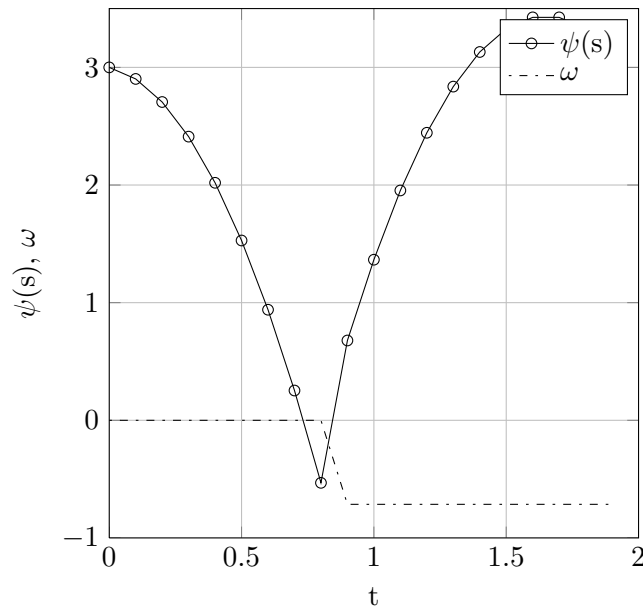
The sphere has an initial horizontal velocity, but no initial rotational velocity. However, after the collision, it starts to rotate. This is due to the torque that is applied by the friction force at the time-step where the sphere is penetrating the plane. If the normal force at the previous time-step was used to define the boundaries on the friction force, this friction force would be zero (since there was no normal force at the time-step prior to the collision).



**Figure 6-5:** Simulation of a bouncing sphere with an initial horizontal velocity with Blender. Time moves horizontal from left to right, and vertical from top to bottom.

## 6-2 Bullet & Blender algorithm structure

One of the strengths of Bullet & Blender is that they can be accessed and expanded through scripts, to include all kinds of elements (e.g. controllers, spring-dampers and sensors). To



**Figure 6-6:** Non-penetration constraint value and angular velocity of a bouncing sphere with initial horizontal velocity.

compute physics, execute scripts and return a rendered frame to the user Blender has three stages, or loops. This structure is illustrated in Figure C-1. These scripts are written in the popular modern programming language Python, a very easy to understand and intuitive programming language, with a strong resemblance to the code used in Matlab.

### 6-2-1 Rendering loop

The top loop is the rendering loop. At every iteration of this loop, a the model is visualized, and shown on the screen. The clock-speed of this loop depends on the clock-speed of the logic loop, but has a maximum of 60 Frames Per Second (FPS) real- or wall-time. That is means that if one rendered frame represents one second in the simulation, Bullet & Blender can simulate up to 60 times faster than real-time. The simulation time represented by one frame can be set setting the physics FPS, which has a maximum of  $10^4$  Hz.

### 6-2-2 Logic loop

Blender allows the user to create interactive games through its built-in game engine, called Blender Game Engine (BGE). In this environment the user can define sensors and controllers (e.g. if this button is pressed, this object moves to the right). These sensors and controllers are evaluated in a logic loop. Blender allows the user to set a setting called '*logic substeps*', with a maximum of 5. This means that for every frame that is generated, the logic loop has run 5 times. The logic loop can also evoke Python modules or scripts. Through these scripts the user can sense and control virtually everything in the simulation, and add custom elements like springs or actuators. It also allows us to write data to a file for later analysis.

### 6-2-3 Physics loop

The lowest layer is the physics loop. At this stage, the world as it is in Blender is passed to its physics engine, Bullet. Bullet detects contacts, handles the reaction forces (i.e. solves the MNCP), and updates the motion of the system through numerical integration. Blender has a settings called '*physics substeps*', in which we can set an integer value from 1 to 5. When this number is for instance set to 5, the physics loop will run 5 times per iteration of the logic loop.

### 6-2-4 Bullet algorithm overview

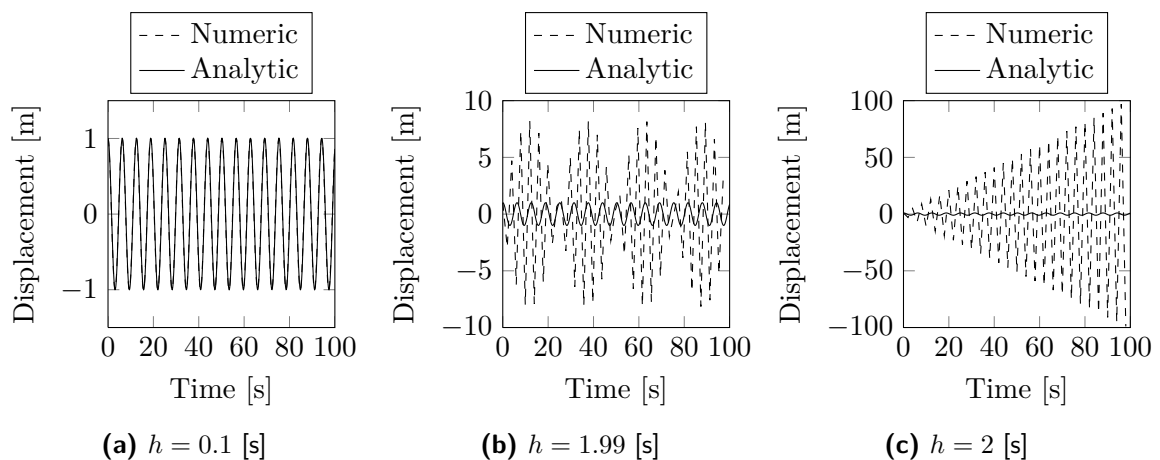
From the research illustrated in this chapter, the structure of Bullet as it is implemented in Blender can at least in part be deduced. This is illustrated in Figure C-2. Note that Blender actually receives the output before any post-stabilization is applied, yielding a larger error in the position shown in Blender.

## 6-3 Inaccuracies

Based on the analysis on the methods used by Bullet & Blender we can discuss what unrealistic behavior we can expect.

### 6-3-1 Semi-implicit Euler integration

The most critical part is the numerical integration method. Semi-implicit Euler (see Appendix B) is a symplectic integrator, and therefore it is said to conserve energy. However, this is not entirely true for all cases. Based on an analysis of how the integrator solves the equation of motion for a simple one-dimensional harmonic oscillator with natural frequency  $\omega$  we identify three cases.



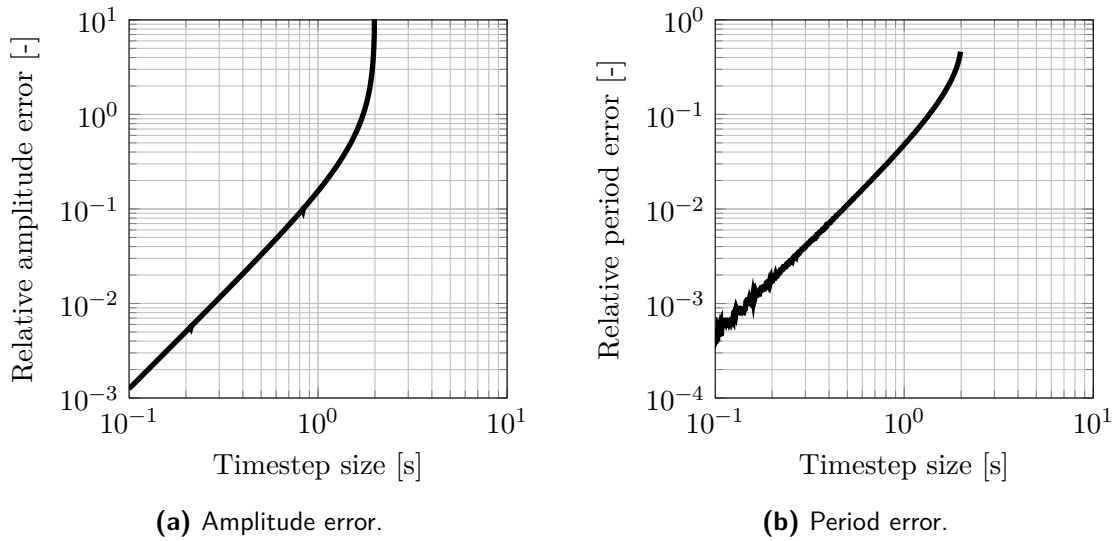
**Figure 6-7:** Analytic versus numerical time response of a harmonic oscillator with  $\omega = 1$  [rad/s] for various time-step sizes.

The first scenario occurs when  $h < \frac{1}{\omega}$ . The amplitude has a small error compared to the analytic solution. However this error does not increase over time and thus integration method conserves energy. The maximum amplitude error is of order  $\mathcal{O}(h^2)$ . This behavior is illustrated in Figure 6-7a.

The second scenario occurs when  $\frac{1}{\omega} < h < \frac{2}{\omega}$ . The signal appears to show AM wave type of behavior, increasing its amplitude periodically far above the analytic amplitude. This behavior is illustrated in Figure 6-7b. The magnitude and period of the amplitude variation increase with  $h$ . However, note that although very inaccurate, the total energy of the system is still not ever increasing, and therefore the integration is not fully unstable.

The final scenario occurs when  $h \geq \frac{2}{\omega}$ . At this point the energy of the system does keep increasing indefinitely, and the integration is fully unstable. This is illustrated in figure Figure 6-7c.

Simulating a large number of oscillations (i.e.  $> 10^3$ ) for a range of time-step sizes and taking the maximum absolute value of the signal, the relation between time-step size and amplitude error can be plotted on a logarithmic scale Figure 6-8a. For  $h < \frac{1}{\omega}$  the error increases with  $\mathcal{O}(h^2)$ , and after that the order increases rapidly until the integration has become unstable. The period error was examined by comparing the frequency of the highest peak in the frequency spectrum of the signal to the analytic frequency of the oscillator.<sup>2</sup> This is shown in Figure 6-8b. Again the error is  $\mathcal{O}(h^2)$  when  $h < \frac{1}{\omega}$ , but increases rapidly beyond that point.



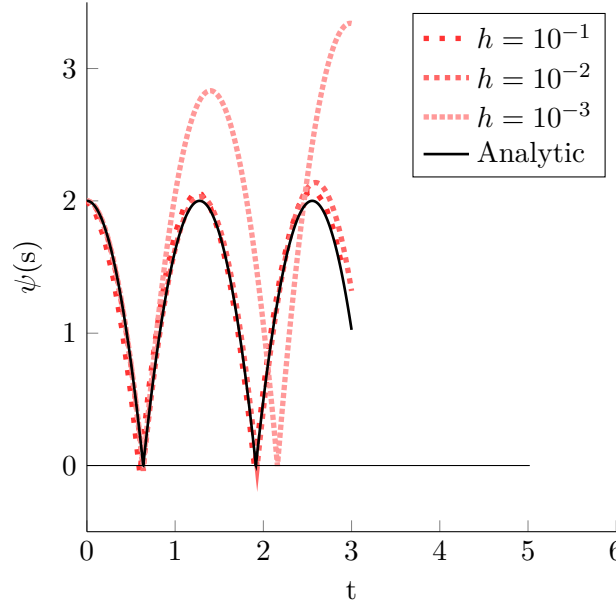
**Figure 6-8:** Amplitude and period error of the time-integration of a harmonic oscillator relative to analytic amplitude and period.

### 6-3-2 Collisions

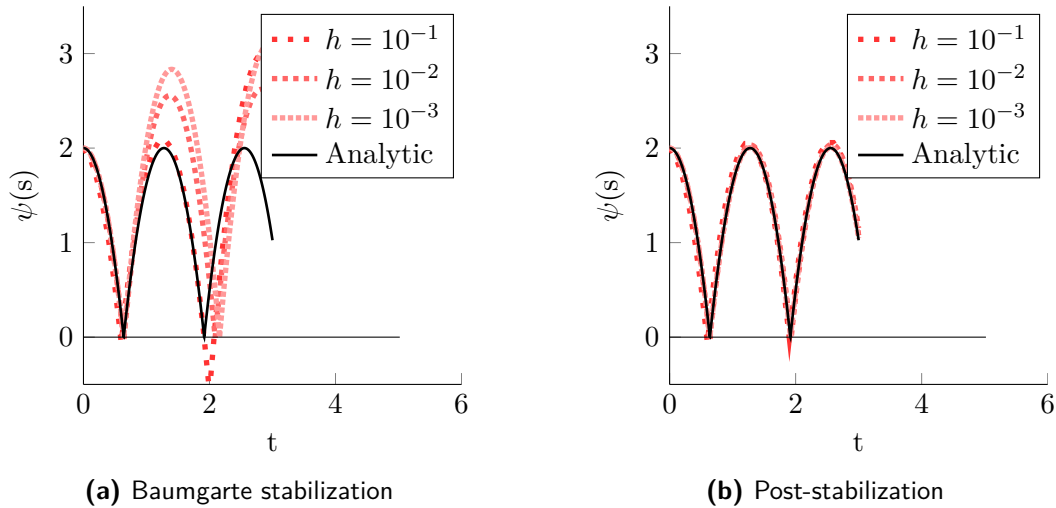
The most obvious source of unphysical behavior occurs in elastic collisions. A fully elastic collision of a sphere on a static surface (i.e.  $\epsilon = 1$ ) results in the sphere maintaining its

<sup>2</sup>Note that a technique called zero-padding was applied to increase the resolution of the Fourier analysis.

energy level exactly. However, as seen in Figure 6-9 in Bullet & Blender the energy seems to increase. Due to the relation between the time-step size and the ‘stiffness’ of the Baumgarte error reduction term (Eq. (4-44)) this behavior is not reduced by choosing a smaller time-step, and therefore the solution does not converge if Baumgarte stabilization is used (Figure 6-10a). Post-stabilization is independent of the time-step size and does converge Figure 6-10b. However, since Bullet & Blender seem to use a mix of both methods, energy will be created in collisions, and this effect behaves pseudo-random and does not converge to the analytic solution of Newton’s hypothesis (see section 4-5).



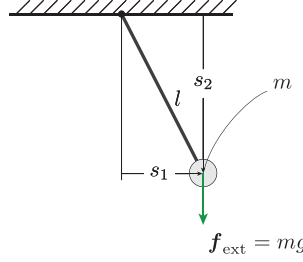
**Figure 6-9:** Value of the non-penetration constraint  $\psi(s)$  of a bouncing sphere over time simulated in Blender for different time-step sizes.



**Figure 6-10:** Value of the non-penetration constraint  $\psi(s)$  of a bouncing sphere over time simulated in Matlab with Baumgarte stabilization and post-stabilization for different time-step sizes. For post-stabilization  $k_{erp} = 0.8$  and for Baumgarte  $k_{erp} = 0.2$ .

### 6-3-3 Joint stabilization

Another field where we see unphysical behavior is in handling bilateral constraints. This can be illustrated by analyzing a model of a single pendulum.



**Figure 6-11:** A single pendulum. Note that although the arrow corresponding to  $s_2$  is pointing down, the positive direction is up, and the position of the hinge is the origin.

Our pendulum is modeled as a point mass with two degrees of freedom ( $s_1$  and  $s_2$ ). It is connected to the hinge by an un-deformable link with length  $l$ . The constraint on the point-mass is

$$\phi(\mathbf{s}) = s_1^2 + s_2^2 - l^2 = 0, \quad (6-2)$$

and therefore our Jacobian reads

$$\mathbf{J}_e = [2s_1, 2s_2]. \quad (6-3)$$

We want to find the reaction impulse, defined as

$$\mathbf{p}_e = \mathbf{J}_e^T \lambda_e. \quad (6-4)$$

From Eq. (4-36) we eliminate all terms related to friction and inequality constraints and append the Baumgarte stabilization. Rewriting explicitly for  $\lambda_e$  yields

$$\mathbf{J}_e \mathbf{M}^{-1} \mathbf{J}_e^T \lambda_e = -\mathbf{J}_e \left( \mathbf{u}^n + h \mathbf{M}^{-1} \mathbf{f}_{\text{ext}} \right) - \frac{k_{\text{erp}}}{h} \phi(\mathbf{s}^n), \quad (6-5)$$

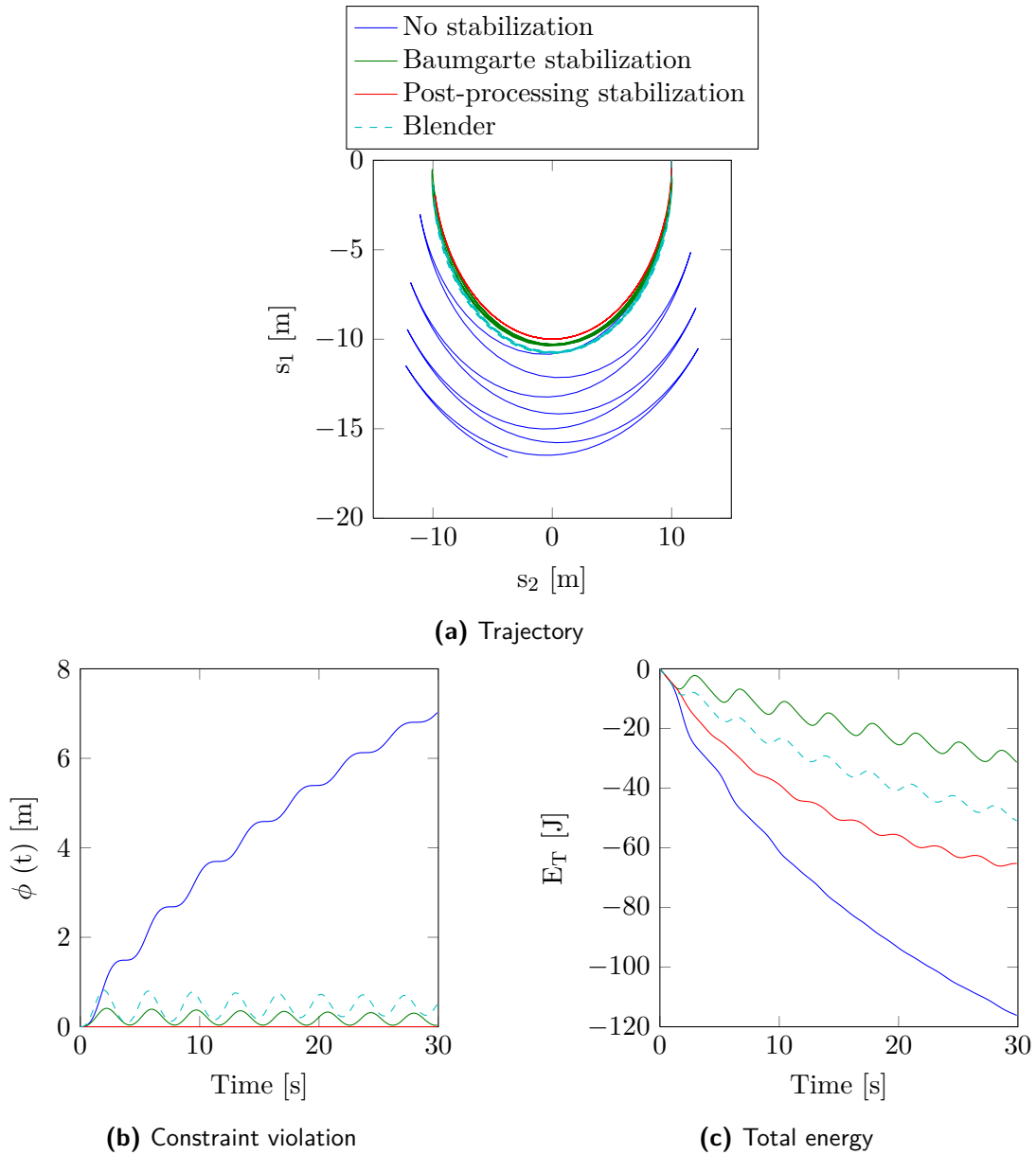
which can be solved for  $\lambda_e$ , which can in turn be plugged into the discrete equation of motion. Furthermore, the total energy of this system is defined as the sum of its kinetic energy (i.e.  $\mathcal{T}$ ) and its potential energy (i.e.  $\mathcal{V}$ ), such that

$$E_T = \mathcal{T} + \mathcal{V} = \frac{1}{2} m (u_1^2 + u_2^2) + mgs_2. \quad (6-6)$$

Figure 6-12 shows trajectories, error and total energy for simulations performed with Matlab implementations with no constraint stabilization, Baumgarte stabilization and post-stabilization, and a simulation performed with Bullet & Blender. In the exact solution, the trajectory of the point-mass would describe half of a circle in Figure 6-12a. The constraint violation would be zero all the time, and the total energy would remain constant.

With no stabilization very large joint errors arise almost immediately. The absolute violation of the constraint is least when using post-stabilization. However, this method also yields





**Figure 6-12:** Results of simulating a single pendulum with Matlab and Blender. Plots indicate position in the 2D space, violation of the constraint over time, and total energy level over time. For these simulations a time-step of  $h = 0.1[s]$  was taken. For the Baumgarte stabilization method, the error reduction parameter was set to  $k_{erp} = 0.2$ .

relatively high energy loss over time (note that no damping has been added to the system). With every swing, the maximum height at the end of swing decreases.

With Baumgarte stabilization the mass oscillates at the end of the link, as if a spring were attached to it. The error is larger than that of the post-stabilization method, but not increasing over time.

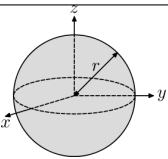
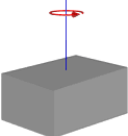
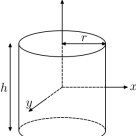
Bullet & Blender show similar behavior as the implementation of Baumgarte error reduction, but not exactly the same. A possible explanation for this could be the fact that in Bullet the

mass at the end of the link is not a point-mass, but has it's own inertia.

### 6-3-4 Inertia tensor computation

Blender computes the inertia tensor of an object based on several primary shapes (i.e. sphere, cube, cylinder) or based on the triangular mesh of the object, and passes this tensor to Bullet. However, computation of the tensor from a arbitrary mesh is not very accurate, and does not converge with the quality of the mesh.

An inertia tensor is a  $3 \times 3$  matrix, denoted as  $\mathbf{I}$ , that relates torque applied to change in angular velocity described in a certain orthogonal basis. However, for every shape we can orient this basis in such a way that the non-diagonal terms in the inertia tensor are zero. The axes of this particular basis are called the principle axes. Bullet describes an inertia tensor of an object with respect to its principle axes, and stores it as a vector containing the three diagonal entries. We will look at three different elementary shapes, listed in table 6-1.

Shape	Figure	Inertia tensor
Sphere		$\mathbf{I} = \frac{2}{5}mr^2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Cuboid		$\mathbf{I} = \frac{1}{12}m \begin{bmatrix} (h^2 + d^2) & 0 & 0 \\ 0 & (w^2 + d^2) & 0 \\ 0 & 0 & (w^2 + h^2) \end{bmatrix}$
Cylinder		$\mathbf{I} = \frac{1}{12}m \begin{bmatrix} (3r^2 + h^2) & 0 & 0 \\ 0 & (3r^2 + h^2) & 0 \\ 0 & 0 & 6r^2 \end{bmatrix}$

**Table 6-1:** List of inertia tensors along principle axis of elementary shapes.

Blender allows to scale the inertia tensor, using a so called scaling factor. This can be interpreted as a measure for mass distribution in the object. Larger numbers relate to mass being distributed more on the edge of the body.

Finally, Blender has a selection box that reads ‘compound’. This affects the inertia tensor, although it is unclear in which way.

**Sphere** For a solid sphere with radius of one meter and a mass of one kilogram, according to table 6-1, an inertia tensor described with respect to the principle axes should yield a matrix with diagonal entries  $[0.4, 0.4, 0.4]$ . Table 6-2 shows the inertia tensor as computed by Blender for a sphere (meshed as an ico-sphere).

Collision bound	Compound	Form factor	Blender output
Sphere	No	1.0	[1, 1, 1]
Sphere	Yes	1.0	[1.67, 1.67, 1.67]
Convex hull	No	1.0	[1.67, 1.59, 1.59]
Convex hull	Yes	1.0	[1.67, 1.59, 1.59]
Triangle mesh	No	1.0	[1.67, 1.67, 1.67]
Triangle mesh	Yes	1.0	[0.67, 0.59, .59]

**Table 6-2:** Blender computed diagonal terms of the inertia tensor of an ico-sphere for various collision bounds. Exact values for a sphere with evenly distributed mass are [0.4, 0.4, 0.4].

**Cuboid** For a solid cuboid with a width, length and height of one meter and a mass of one kilogram, according to table 6-1, an inertia tensor described with respect to the principle axes should yield a matrix with diagonal entries of approximately [0.167, 0.167, 0.167]. Table 6-3 shows the inertia tensor as computed by Blender for a box.

Collision bound	Compound	Form factor	Blender output
Box	No	1.0	[0.417, 0.417, 0.417]
Box	Yes	1.0	[0.417, 0.417, 0.417]
Convex hull	No	1.0	[0.417, 0.417, 0.417]
Convex hull	Yes	1.0	[0.417, 0.417, 0.417]
Triangle mesh	No	1.0	[1.25, 1.25, 1.25]
Triangle mesh	Yes	1.0	[0.417, 0.417, 0.417]

**Table 6-3:** Blender computed diagonal terms of the inertia tensor of an cuboid for various collision bounds. Exact values for a cuboid with evenly distributed mass are [0.167, 0.167, 0.167].

**Cylinder** For a solid cylinder with a height of two meters and a radius of one meter, according to table 6-1, an inertia tensor described with respect to the principle axes should yield a matrix with diagonal entries of approximately [0.58, 0.58, 0.5]. Table 6-3 shows the inertia tensor as computed by Blender for a box.

Collision bound	Compound	Form factor	Blender output
Cylinder	No	1.0	[1.46, 1.46, 1.25]
Cylinder	Yes	1.0	[1.67, 1.67, 1.67]
Convex hull	No	1.0	[1.67, 1.67, 1.67]
Convex hull	Yes	1.0	[1.67, 1.67, 1.67]
Triangle mesh	No	1.0	[3.75, 3.75, 2.5]
Triangle mesh	Yes	1.0	[1.67, 1.67, 1.67]

**Table 6-4:** Blender computed diagonal terms of the inertia tensor of a cylinder for various collision bounds. Exact values for a cylinder with evenly distributed mass are [0.58, 0.58, 0.5].

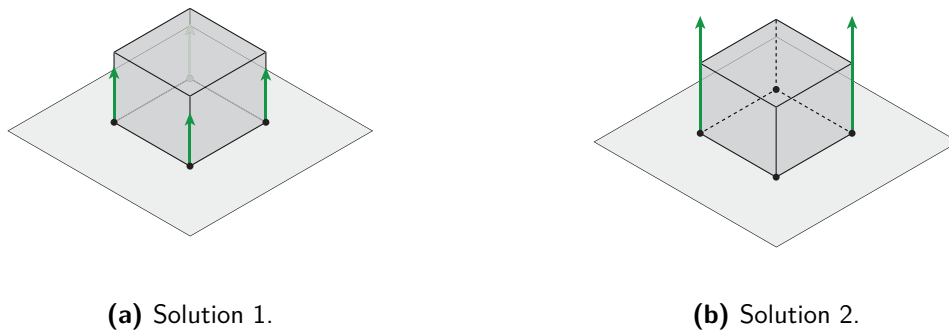
**Comparison** We can see that simple shapes are best represented by using elementary collision bounds, if we use the proper scaling. However, even for these simple shapes, obtaining a correct inertia tensor from a meshing is not trivial, and sometimes even impossible. It is therefore highly unlikely that Blender will be successful in computing the inertia tensor of more complex shapes. Bullet offers a way to user-define the inertia tensor of a body, but this is not exposed in the current version of Blender.

### 6-3-5 Solution convergence and uniqueness

One of the characteristics of the PGS algorithm, is that for the first 5-10 iterations it has a high order of convergence, but the convergence flattens out when the number of iterations increases. Therefore it is relatively good at finding a first approximation of a problem, but not good at finding high accuracy solutions[37].

As stated before, the solver continues to iterate the problem, until either certain convergence criteria are met, or a maximum number of iterations is reached. A convergence criterion can be some complicated error term, which requires time to compute every iteration. For applications where speed is more important than control over the error it is therefore suggested to opt for a fixed number of iterations, and not check the error in the problem. This is also what has been applied in Bullet. The problem with this is that the user cannot control or measure the magnitude of the error in the computation of the reaction forces.

Several methods exist, such as subspace minimization that improve the convergence of the PGS algorithm, but it is unclear whether these have been applied to Bullet & Blender.



**Figure 6-13:** Illustration of the lack of uniqueness in the solution of a statically undetermined system.

Another characteristic of the contact problem used in Bullet is that although solution existence has been guaranteed, solution uniqueness is not. Consider a block with evenly distributed mass, subject to gravity, resting on a table. In this case we can choose to model the contact plane as four contact points at the corners of the block, as illustrated in Figure 6-13. The most plausible solution to this problem would be to spread the contact forces equally over all four contact points of the block. However, another solution would be to spread the complete reaction force over two of the contact points, and leave the other two unloaded. Hence, there is no unique solution to this problem if we use constraint based simulation.

However, this lack of uniqueness is inherent to the assumption of a rigid-body. Without taking the elastic properties of the block into account, the problem is statically undetermined. In

some literature it is suggested that the solution will tend to an equal distribution of the contact forces, but there exists no clear proof of such a statement.



## **Part II**

# **Experimental verification**





---

## Chapter 7

---

# Elastic collisions

This chapter discusses the experimental verification of elastic collisions, in this case a tennis ball bouncing on a flat surface. The first sections of this chapter will discuss the test setup. The second part will compare the results to simulations, both penalty based and constraint based methods. The third part will compare simulations performed with Bullet & Blender to more established tools.

### 7-1 Test setup

The tests have been performed by dropping a tennis ball on the floor. The properties of the tennis ball are noted in Table 7-1. A black backdrop was used to facilitate the image processing. The position was recorded using a Sony high-speed camera, shown in Figure 7-1, fixed on a tripod. The position of the ball was extracted using an image processing algorithm programmed in Matlab. For more on this algorithm, the reader is referred to Appendix A.

In initial tests with a GoPro wide-angle camera showed that the deformation of the image by the lens affected the accuracy of the tracking. Therefore it was decided to set the camera as far back as possible ( $\simeq 6$  [m]). To make sure the ball was dropped from the same point every time a construction made of two tripods and a block of wood was used. Figure 7-2 shows a frame from one of the recordings, illustrating the test setup.

Property	Value
Mass	0.0548 [kg]
Diameter	0.0335 [m]

**Table 7-1:** Tennis ball properties



**Figure 7-1:** Sony NEX FS-700 high-speed camera[40].



**Figure 7-2:** Screenshot from recording showing the test setup for the tennis ball experiments.

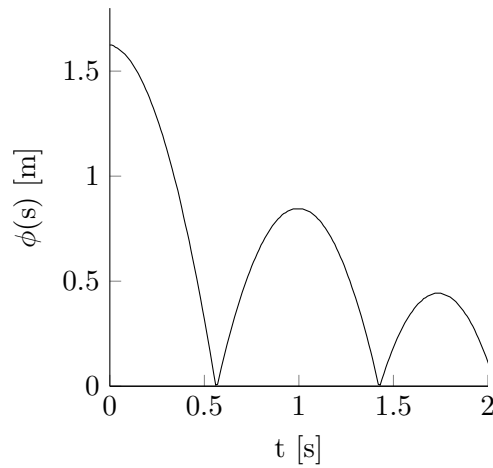
## 7-2 Experimental and simulated results

As can be seen in appendix A the tracking is very accurate in finding the position of the ball. The corresponding position data are plotted in Figure 7-3. To get an idea of the spread of the data, the experiment was repeated several times. The results of all these experiments are plotted on the same axis in Figure 7-4. We can see that the repeatability of this experiment is in the order of several centimeters.

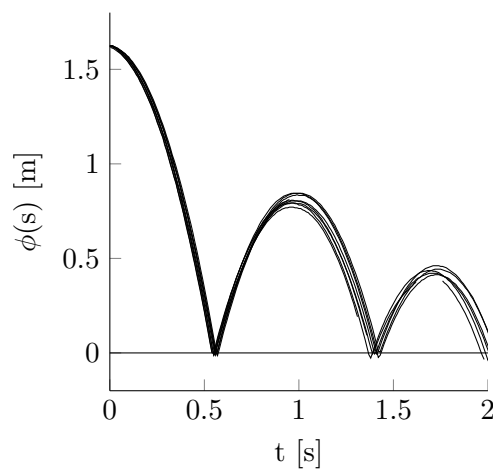
Apart from gravity, the motion of the tennis ball is also affected by aerodynamic drag. The magnitude of this force is commonly computed through the drag equation (Eq. (8-19)).

$$\mathbf{F}_d = -\frac{1}{2}AC_D\rho(\mathbf{u} \cdot \mathbf{u})\left(\frac{\mathbf{u}}{\|\mathbf{u}\|}\right) \quad (7-1)$$

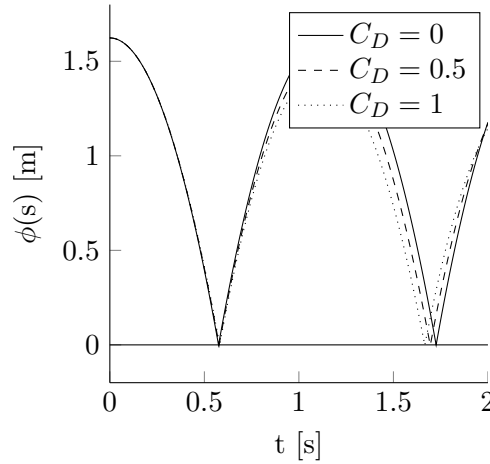
Here  $\mathbf{F}_d$  represents the friction force,  $A$  is the cross-section reference area of the object perpendicular to the motion,  $\rho$  the density of the flow medium and  $C_D$  represents the experimentally determined drag coefficient.



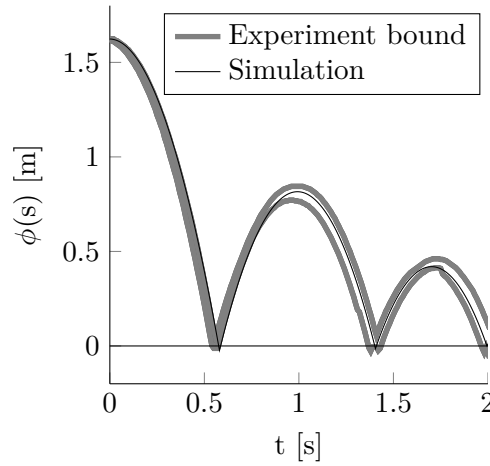
**Figure 7-3:** Tracked position of tennis ball over time. Note that the shown position is the height of the bottom of the ball with respect to the ground which can be interpreted as the value of the non-penetration constraint when comparing to constraint based simulations.



**Figure 7-4:** Tracked position of the tennis ball, repeated several times.



**Figure 7-5:** Simulated trajectories of tennis ball, with different drag coefficients. Here  $\epsilon = 1$  and  $C_D = [0, 0.5, 1]$ .



**Figure 7-6:** Maximum and minimum boundaries of experimental data compared to simulated position of the tennis ball. Here  $\epsilon = 0.73$  and  $C_D = 0.5$ .

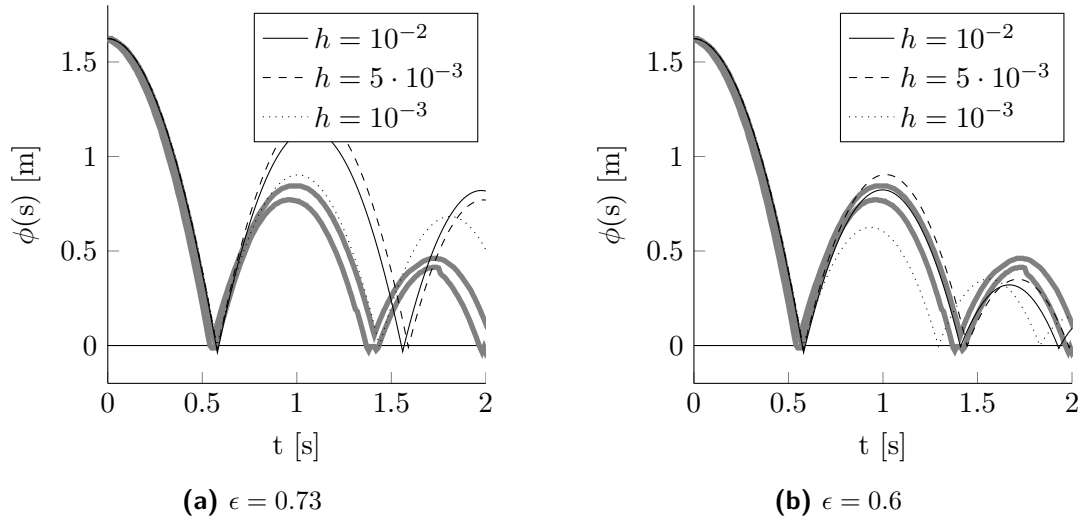
The drag coefficient of a tennis ball varies significantly depending on its orientation and velocity[41]. The effect of drag on the motion of the ball is shown in Figure 7-5. Based on earlier research by Alam, Watkins and Subic on drag coefficients of tennis balls, a value of  $C_D = 0.5$  is selected for all following simulations.

We can determine an approximation for the coefficient of restitution that best represents the experiment by matching the output of a simulation performed with Matlab (without any constraint stabilization) to the experimental data. Simulating with  $\epsilon = .73$  results in the trajectory shown in Figure 7-6, which seems to match the experimental data appropriately.

Simulating this simple experiment with Bullet & Blender for different time-step sizes reveals the pseudo-random behavior in how elastic collisions are handled due to the error reduction. In Figure 7-7a it can be seen that when we keep the same coefficient of restitution as determined before, the exit velocity after a collision is overestimated. In Figure 7-7b the restitution is reduced to better match the experiments. However, here it can be seen that the repeatability

of the simulation is much lower than that of the experiments.

We can conclude that Newton's hypothesis can predict the behavior of tennis ball within the repeatability of the experiment. However, we can conclude that Bullet, due to its error reduction method, is not able to reproduce the experimental results with sufficient accuracy.



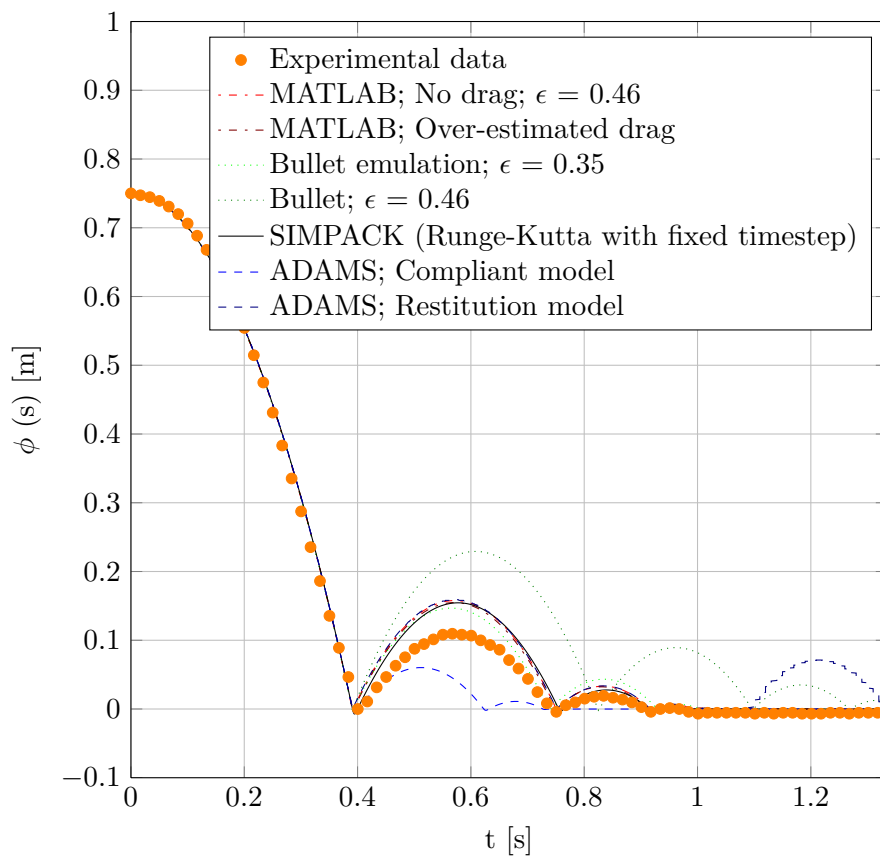
**Figure 7-7:** Simulation of tennis ball with Bullet & Blender for various time-step sizes  $h$  [s].

### 7-3 Comparison to established tools

Apart from experiments with the tennis ball, several other experiments have been performed with a squash ball. This experiment has been compared to several simulations performed with more established tools, such as MSC/Adams and SIMPACK. These results are shown in Figure 7-8.

The first thing we can note is the fact that the second arc of the experiment is considerably lower than that of the simulated trajectories. It was found that this was because the experiment was recorded from close-range with a GoPro wide-angle camera, which distorted the image. Therefore all the tennis ball experiments were recorded from ( $\simeq 6$  [m]), with a normal lens.

The second thing that can be noted is that SIMPACK is more successful than Adams in reproducing the experimental data. Both contact models that can be selected in Adams (*compliant* and *restitution*) are unsuccessful. The first contact model experiences too much artificial damping, and the second model becomes unstable after 1.1 seconds. We can conclude that simulating even a very simple elastic collision with Adams is not trivial at all.



**Figure 7-8:** Position of squash ball from experimental data and simulations performed with Matlab, Blender, SIMPACK and MSC/Adams.

---

## Chapter 8

---

# String dynamics

As discussed in the introduction of this thesis, this project was originally driven by research on capturing orbital debris using throw-nets. A critical component of any net simulation is the elastic behavior of the net. In a rigid-body simulator one can model this using a technique referred to as *lumped-mass modeling*. Here we approximate the flexible body with a large number of rigid-bodies connected through spring-dampers.

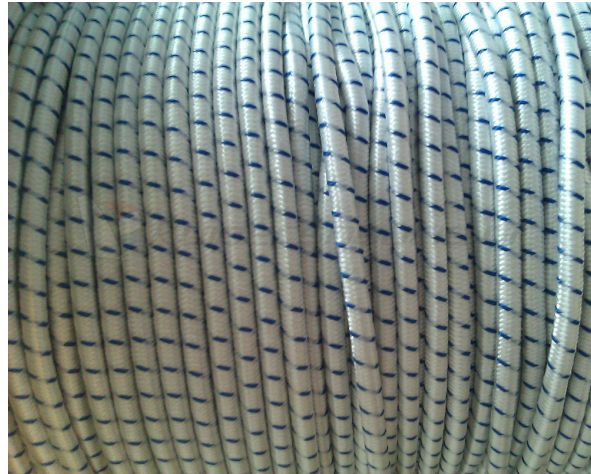
In order to verify this technique, a simple string experiment was performed. The first two sections of this chapter will describe the experiment setup and models used for simulation. Second, the resulting time signal and frequency response of a several excitations are compared to simulations. Finally, stability issues and convergence are discussed.

### 8-1 Test setup

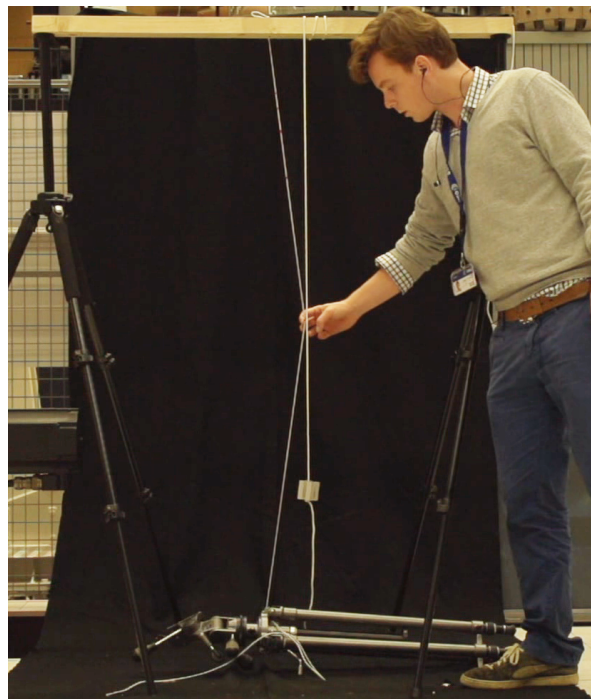
The test setup consists of an elastic cord or string (Figure 8-1) suspended vertically under pretension between a block of wood and a heavy metal object ( $\approx 6$  [kg]). The string was manually brought to a position away from its equilibrium and released. The resulting motion was recorded with the high-speed camera used earlier for the collision trajectory analyses (Figure 7-1). The resulting video was processed in a similar way as the tennis ball experiment to extract the position over time of several points along the string. Characteristics of the string can be found in Table 8-1 and the test setup is shown in Figure 8-2.

Property	Symbol	Value
Linear density	$\rho_{\text{lin}}$	0.0075 [kg / m]
Diameter	$d$	0.003 [m]
Natural length	$l_n$	1.520 [m]
Length under pretension	$l_t$	1.558 [m]

**Table 8-1:** String properties



**Figure 8-1:** The cord used for the experiments



**Figure 8-2:** String experiment test setup



The stiffness of the string was determined by suspending a known probe mass on a piece of the string with a known natural length. By measuring the elongation of the string, the Young's modulus could be estimated using Hooke's law (Eq. (8-1)).

$$\sigma = E\epsilon \quad (8-1)$$

Here  $E$  represents the Young's modulus,  $\sigma$  the normal stress and  $\epsilon$  the strain in the string. By rewriting it in terms of the cords natural length  $l'_n$ , the probe mass  $m'$ , the gravitational constant  $g$ , the string radius  $\rho$  and the new or stretched length of the string  $l'_t$  we obtain an explicit expression for the Young's modulus. This yields

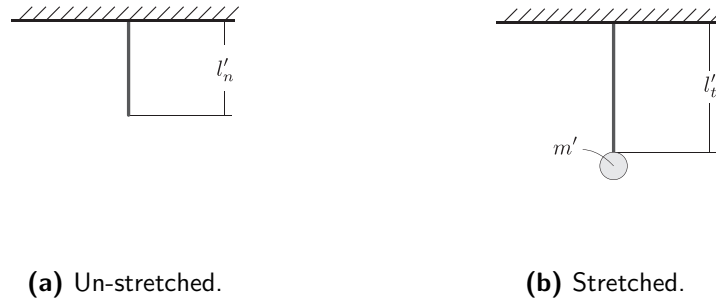
$$\frac{F}{A} = E \left( \frac{l'_t - l'_n}{l'_n} \right), \quad (8-2a)$$

$$E = \frac{Fl'_n}{\pi\rho^2(l'_t - l'_n)}, \quad (8-2b)$$

where

$$F = m'g. \quad (8-2c)$$

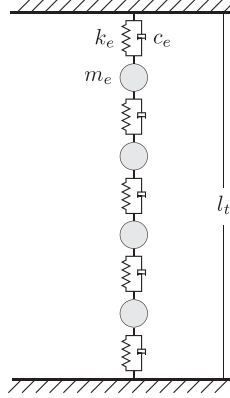
Table 8-2 lists the measured values used for determination of the Young's modulus. The method is illustrated in Figure 8-3.



**Figure 8-3:** Illustration of method used to determine Young's modulus of string.

Property	Symbol	Value
Probe mass	$m'$	0.2118 [kg]
Natural length	$l'_n$	1.315 [m]
Stretched length	$l'_t$	1.351 [m]
String radius	$\rho$	0.0015 [m]
Gravitational acceleration	$g$	9.81 [m/s <sup>2</sup> ]
Young's modulus	$E$	$1.07 \times 10^7$ [Pa]

**Table 8-2:** String properties



**Figure 8-4:** A lumped mass model of a string with four masses.

## 8-2 Model

In a rigid-body simulator all bodies are inherently un-deformable. As discussed before, one way to model flexible bodies such as nets and strings is as a combination of small masses connected with spring-dampers, illustrated in Figure 8-4. The mass of the body is distributed over the masses. As the number of masses increases the behavior of this model should converge to the actual behavior of the body. In this section we discuss how this model is built up.

### 8-2-1 Spring forces

The position and velocity of mass  $i$  are represented by respectively  $\mathbf{r} \in \mathbb{R}^3$  and  $\mathbf{v} \in \mathbb{R}^3$ .

We define an operator  $\delta_{ij}\bullet$  as the entry-wise subtraction of two vectors related to masses  $i$  and  $j$ , such that

$$\delta_{12}\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_2, \quad (8-3a)$$

$$\delta_{12}\mathbf{v} = \mathbf{v}_1 - \mathbf{v}_2. \quad (8-3b)$$

Now  $\delta_{12}\mathbf{r}$  represents a vector from the position of mass 2 to position of mass 1. We define a vector  $\delta_{ij}\hat{\mathbf{r}}$  that represents the unit vector in the direction from the position of mass  $j$  to the position of mass  $i$ , such that

$$\delta_{ij}\hat{\mathbf{r}} = \frac{\delta_{ij}\mathbf{r}}{\|\delta_{ij}\mathbf{r}\|}. \quad (8-4)$$

This allows us to write the magnitude of the spring force between element masses  $i$  and  $j$  as

$$F_{s,ij} = k_e(\|\delta_{ij}\mathbf{r}\| - l_e), \quad (8-5)$$

where  $l_e$  is the natural length of the spring connecting both masses and  $k_e$  represents the element stiffness. Projecting this onto the direction of the spring yields

$$\mathbf{F}_{s,ij} = k_e(\|\delta_{ij}\mathbf{r}\| - l_e)\delta_{ij}\hat{\mathbf{r}}, \quad (8-6)$$

where  $\mathbf{F}_{s,ij}$  represents a force vector pointing from  $\mathbf{r}_j$  to  $\mathbf{r}_i$  if the string is stretched, and from  $\mathbf{r}_i$  to  $\mathbf{r}_j$  if the spring is compressed. Therefore, in the absence of any damping, the spring forces acting on bodies  $i$  and  $j$  are

$$\mathbf{F}_{s,i} = -\mathbf{F}_{s,ij}, \quad (8-7a)$$

$$\mathbf{F}_{s,j} = \mathbf{F}_{s,ij}. \quad (8-7b)$$

### 8-2-2 Damping forces

Viscous damping can be added in similar fashion. The damping force acts in the same direction as the spring force (i.e. along  $\delta_{ij}\hat{\mathbf{r}}$ ). The magnitude of the velocity along this direction can be found by projecting the relative velocity of  $i$  and  $j$  onto the direction of the spring damper. Multiplying this relative velocity with an element damping coefficient  $c_e$  yields the magnitude of the damping force, written as

$$F_{d,ij} = c_e(\delta_{ij}\mathbf{v} \cdot \delta_{ij}\hat{\mathbf{r}}). \quad (8-8)$$

By again projecting this magnitude in the direction of the spring-damper we obtain the damping force in vectorial form.

$$\mathbf{F}_{d,ij} = c_e(\delta_{ij}\mathbf{v} \cdot \delta_{ij}\hat{\mathbf{r}})\delta_{ij}\hat{\mathbf{r}} \quad (8-9)$$

If  $i$  and  $j$  are approaching each other,  $\mathbf{F}_{d,ij}$  results in a vector pointing from  $\mathbf{r}_i$  to  $\mathbf{r}_j$ , and if the bodies are moving away from each other it results in a vector pointing from  $\mathbf{r}_j$  to  $\mathbf{r}_i$ . Therefore the damping forces acting on the bodies are

$$\mathbf{F}_{d,i} = -\mathbf{F}_{d,ij}, \quad (8-10a)$$

$$\mathbf{F}_{d,j} = \mathbf{F}_{d,ij}. \quad (8-10b)$$

Combining this with Eq. (8-7) yields the total forces on the bodies due to the spring-dampers.

$$\mathbf{F}_{ij} = \mathbf{F}_{s,ij} + \mathbf{F}_{d,ij}, \quad (8-11a)$$

$$\mathbf{F}_i = -\mathbf{F}_{ij}, \quad (8-11b)$$

$$\mathbf{F}_j = \mathbf{F}_{ij}. \quad (8-11c)$$

### 8-2-3 Slack

In reality stiffness and damping are not linear with respect to the displacement in a string for several reasons. One of those reasons is slack. A string is not able to provide compressive force as an ideal spring would. In an attempt to account for this behavior a discrete model for the spring forces has been used. This model is described by

$$\mathbf{F}_{ij} = \begin{cases} \mathbf{F}_{s,ij} + \mathbf{F}_{d,ij} & \text{if } \|\delta_{ij}\mathbf{r}\| - l_e \geq 0 \\ 0 & \text{if } \|\delta_{ij}\mathbf{r}\| - l_e < 0 \end{cases}. \quad (8-12)$$

Effectively this sets all spring-damper forces to zero if the element is compressed.

### 8-2-4 Element properties

Element properties (i.e. stiffness, mass, length and damping) vary with the number of mass elements used to model the string, defined as  $N$ . The natural length  $l_e$  of any element is defined as

$$l_e = \frac{l_n}{N+1} \quad (8-13)$$

where  $l_n$  represents the natural length of the part of the string between the clamped ends. Element mass  $m_e$  follows from this and the linear density, such that

$$m_e = l_e \rho_{\text{lin}} \quad (8-14)$$

By using the derivation of Hooke's law in section 8-1, writing Eq. (8-2a) for one element the stiffness of one element can be defined. We write

$$F_{s,ij} = \frac{EA}{l_e} (||\delta_{ij} \mathbf{r}|| - l_e), \quad (8-15)$$

and by comparing Eq. (8-15) with the expression for spring force magnitude (Eq. (8-5)) we obtain

$$k_e = \frac{EA}{l_e}. \quad (8-16)$$

The element damping coefficient of an element can be related to the element stiffness through the critical damping ratio. For a simple harmonic oscillator it is defined as

$$\zeta = \frac{c}{2\sqrt{mk}} \quad (8-17)$$

For the element damping in the lumped string we define

$$c_e = 2\zeta\sqrt{m_e k_e}, \quad (8-18)$$

where  $\zeta$  will be set to match the experimental behavior.

### 8-2-5 Drag

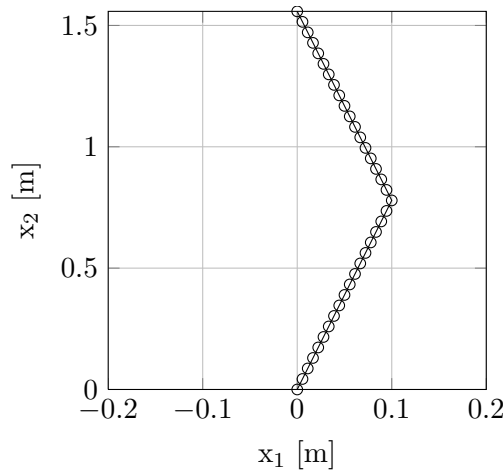
Aerodynamic drag is added to each element in the horizontal direction only. We denote the drag acting on element  $i$  in vectorial form as  $\mathbf{F}_{D,i}$ . It is defined as

$$\mathbf{F}_{D,i} = \frac{1}{2} \rho_{\text{air}} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{v}_i C_D A_h, \quad (8-19)$$

where  $\rho_{\text{air}}$  represents the density of air,  $C_D$  a drag coefficient, and  $A_h$  the cross-sectional area of one mass element perpendicular to the horizontal motion, given by

$$A_h = \frac{l_t d}{N}. \quad (8-20)$$

The  $2 \times 2$  matrix in Eq. (8-19) ensures that the drag only acts in horizontal direction.



**Figure 8-5:** Implementation of the tether in Matlab, shown at initial time  $t_0$ .

### 8-2-6 Implementation in Matlab, Blender & Adams

To analyze the use of lumped mass models in Bullet & Blender, three implementations were made (in Matlab, Blender & Adams).

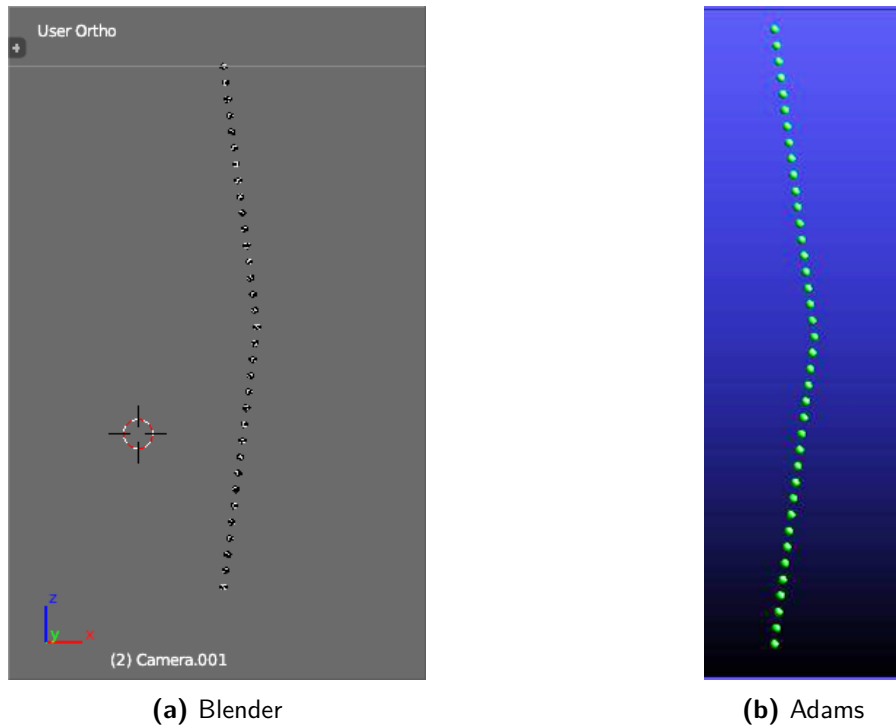
In Matlab a function based implementation was created with the semi-implicit Euler integration method at the heart. Each mass has two translational degrees of freedom and no rotational degrees of freedom. The initial position was set to a triangular shape depending on where the string was excited, neglecting the effect of gravity on the equilibrium. An example of the initial position of the nodes in Matlab is shown in Figure 8-5.

In Bullet & Blender the model has been implemented through two scripts. The first script is run only once to set up the mesh and define parameters simulation parameters. The second script runs every iteration of the Blender Game Engine (BGE) and computes spring and damping forces, updates velocities and creates an output file. All spring forces are applied to the Centre Of Mass (COM) of the masses, and therefore the masses only translate and do not rotate. A screen shot of the implementation in Blender is shown in Figure 8-6a.

Finally, the system is implemented in MSC/Adams. Here all masses and spring-dampers had to be defined individually, making the process slow and tedious, and simulation less flexible. Adams does allow for modeling through scripts, but this was not explored for this thesis. All rotational degrees of freedom of the masses had to be fixed, since they caused instabilities in the simulation, even though the spring and damping forces act on the COM of the masses. A screen shot of the implementation in Adams is shown in Figure 8-6b.

## 8-3 Experimental & simulated results

To facilitate the comparison process, all simulations in this section have been performed with Matlab. Based on the fact that Bullet uses the semi-implicit Euler integration method, and that no contact behavior is present in the simulation of the tether, it is assumed that Bullet will yield the same results.



**Figure 8-6:** Implementation of the lumped tether in model Blender and Adams.

### 8-3-1 Experimental results

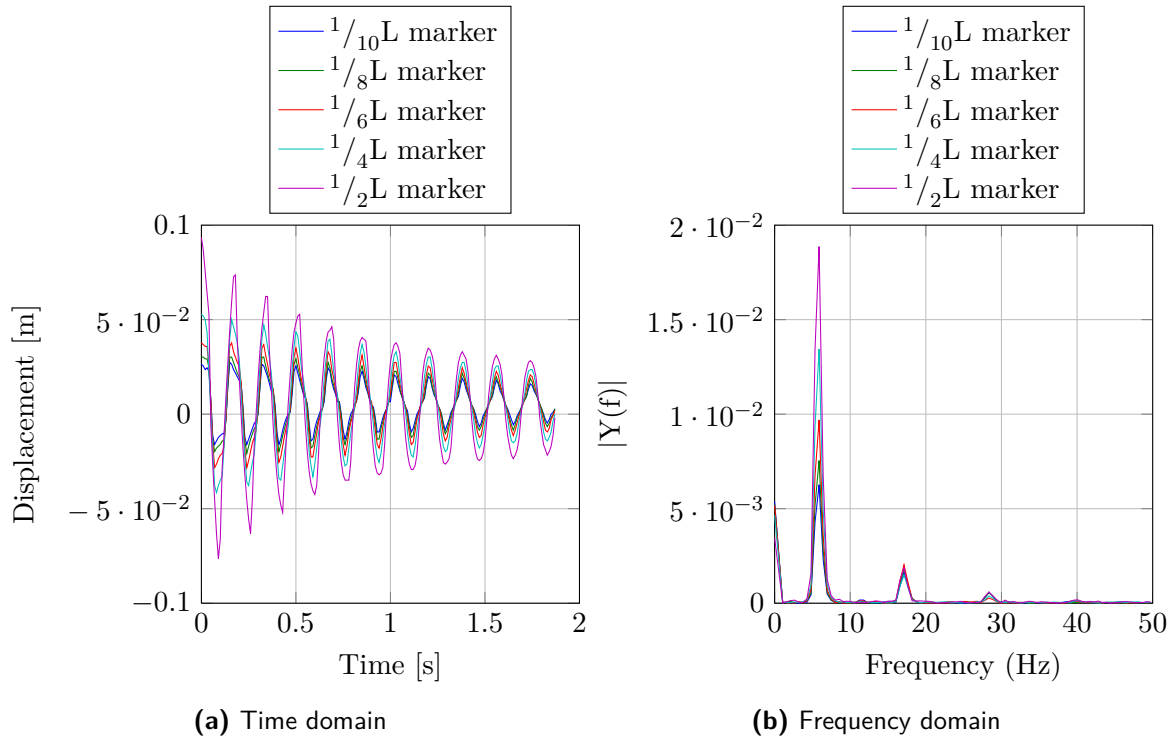
The resulting time signal and corresponding Fourier analysis of the tether excited in a symmetric triangular shape as depicted in Figure 8-2 are shown in Figure 8-7. As can be expected the tether shows a combination of several modes, most clearly visible in the first half of the time signal. The sharp, almost triangular wave is a combination of the first, third and fifth eigenmode of the string, as seen in the frequency domain.

Also, the tether shows a fair amount of non-linear behavior. This is mainly visible after the higher modes have damped out in the second half of the time signal, where the markers describe a stretched sinusoidal motion.

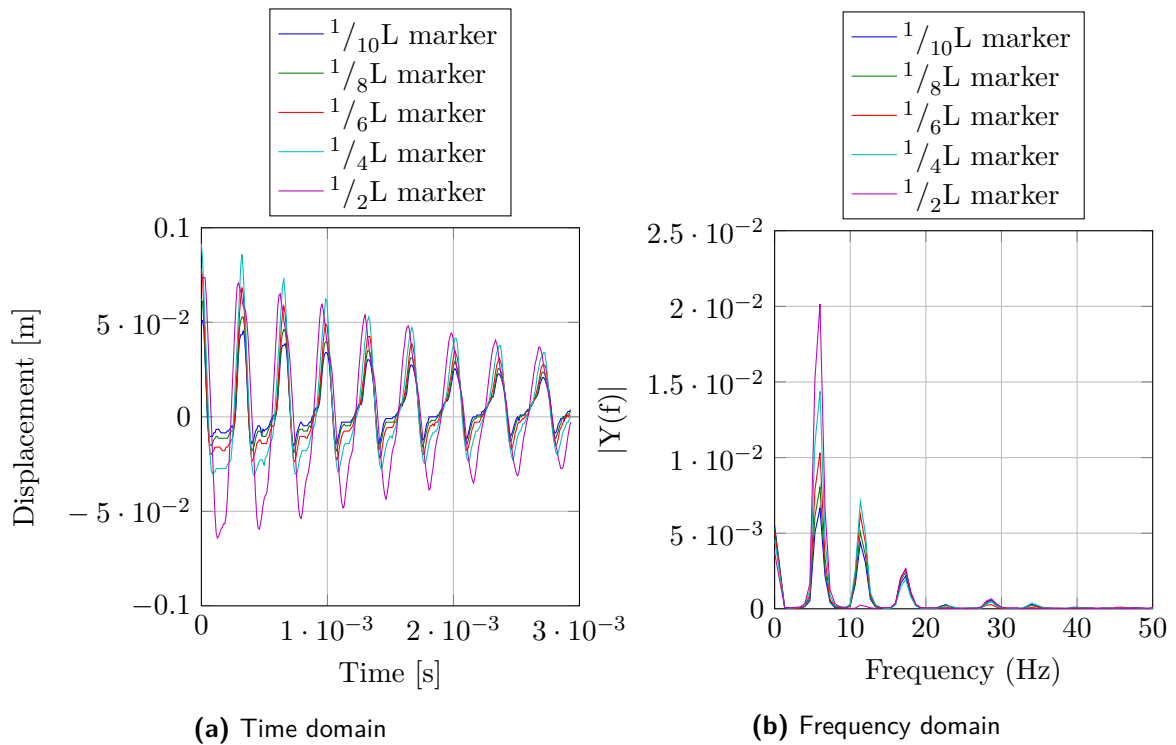
In the frequency domain we also see a peak at the origin. This corresponds to a rigid body mode, or rather, what is considered the centerline of the string in the analysis actually contains an offset with respect to the actual centerline.

Finally, it can also be seen that only odd (i.e. symmetric) eigenmodes are present in the signal (i.e. 1<sup>st</sup>, 3<sup>rd</sup>, 5<sup>th</sup>). This is due to the fact that the triangular shaped initial position is symmetric.

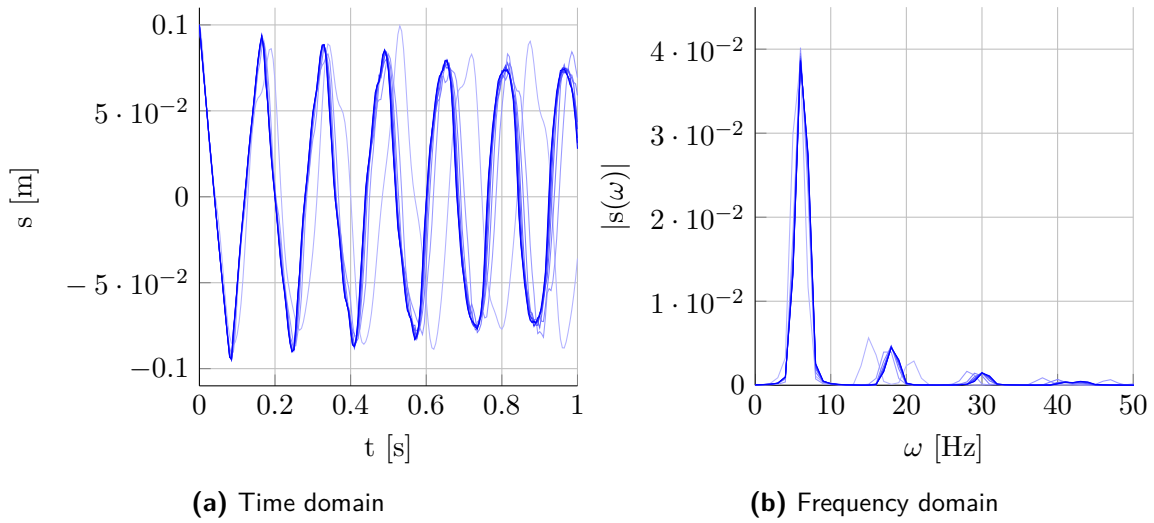
Figure 8-8 shows the time signal and corresponding Fourier transform of the string when excited in a non-symmetric way. Here the string was excited by grabbing and releasing a point at  $\frac{1}{4}L$  distance from the top, instead of at the middle. This initial position is not symmetric, and hence even (i.e. asymmetric) modes are present in the resulting signal.



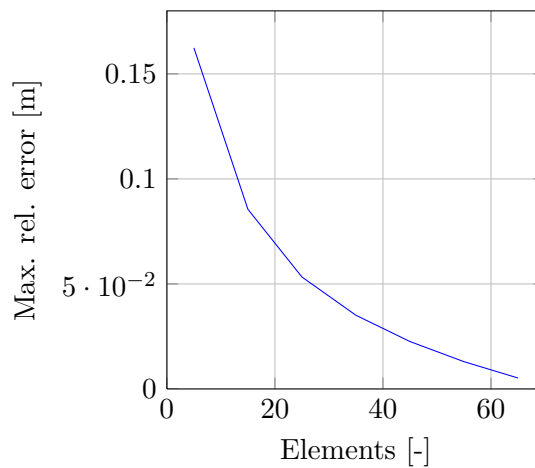
**Figure 8-7:** Time signal of the horizontal displacement of 5 markers along the top half of the string, and respective spectral analysis. String is excited at the  $\frac{1}{2}L$  marker.



**Figure 8-8:** Time signal of the horizontal displacement of 5 markers along the top half of the string, and respective spectral analysis. String is excited at the  $\frac{1}{4}L$  marker.



**Figure 8-9:** Time signal of several simulations in which the string is modeled with increasing number of elements. Darker means more elements, ranging from 5 to 75 elements.



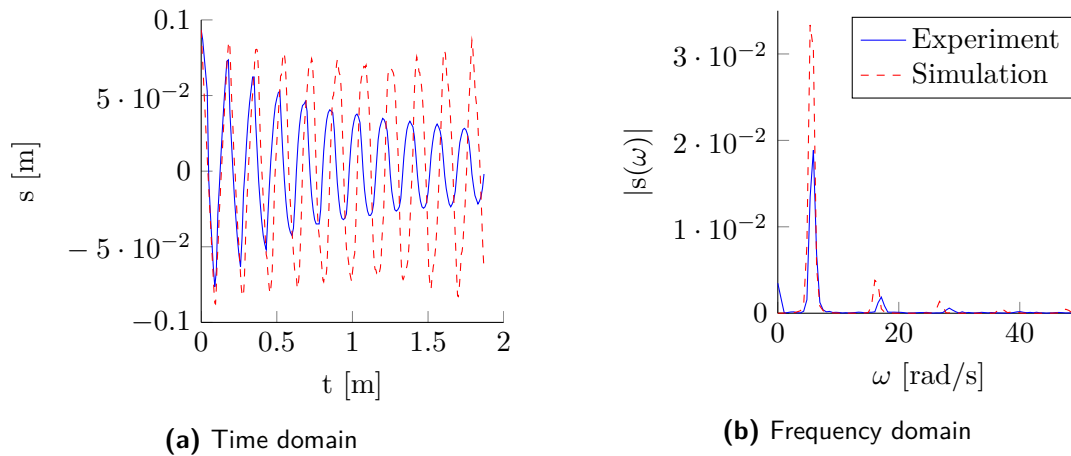
**Figure 8-10:** Maximum error of the simulation, relative to solution with 75 nodes.

### 8-3-2 Number of mass elements

To find the minimal number of elements needed to model the string, an undamped string was simulated with varying number of elements, ranging from 5 to 75. When we overlay the time-signals and Fourier analyses (Figure 8-9) of all simulations we see a clear convergence. Also, plotting the maximum absolute difference between the time-signal and the time-signal of the most complex model (i.e. with 75 elements) shows convergence, as illustrated in Figure 8-10.

Based on this it was decided that any other string simulations described in this report would be performed with models ranging between 30 and 40 elements. The reason that the number of elements used varies is that different points along the line were excited and it is preferable to have a node at the point where the string was excited.





**Figure 8-11:** Time signal and frequency content of middle node displacement for symmetric excitation, experiment versus undamped simulation ( $\zeta = 0, C_D = 0$ ).

### 8-3-3 Comparison

Experimental data of a symmetrically excited string and the corresponding simulation of an un-damped string is shown in Figure 8-11. Here  $C_D = 0$ , stiffness and mass are exactly as measured. We see that behavior is similar but the system is not damped correctly, and the modeled eigenfrequencies are slightly too low.

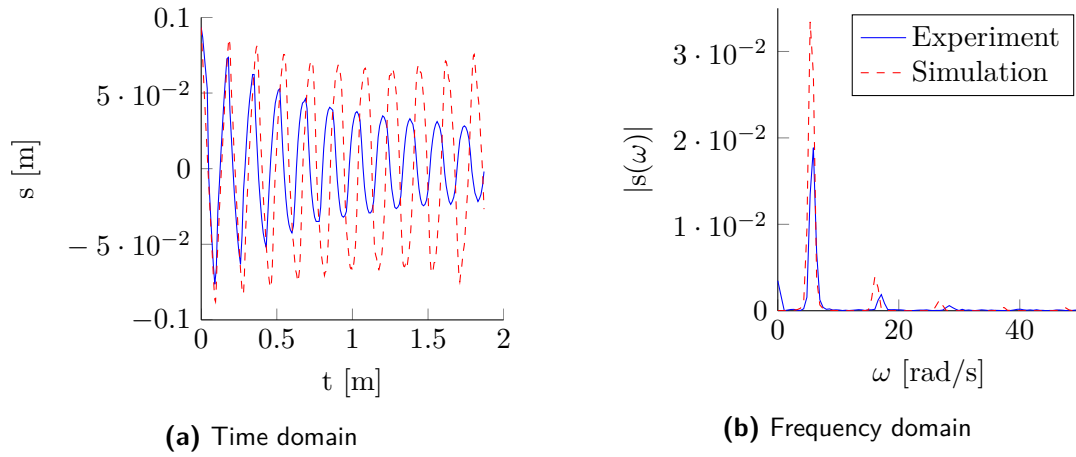
In Figure 8-12 the results for a simulation where the internal damping in the string is increased by setting  $\zeta = 1$  are shown. It can be seen that the damping of the transverse modes is only slightly increased, although computational time to simulate the system without creating unstable higher modes is increased drastically. Apparently the internal damping has very little effect on the transverse motion of the string.

However, including aerodynamic drag by setting  $C_D = 2$  yields a more realistic result, as depicted in Figure 8-13. Since the eigenfrequencies are still too low, the element stiffness has been corrected by a factor of 1.2. This yields the result shown in Figure 8-14, where the simulation and experiment match very accurately. Again, increasing the critical damping ratio has very little effect, as seen in Figure 8-15.

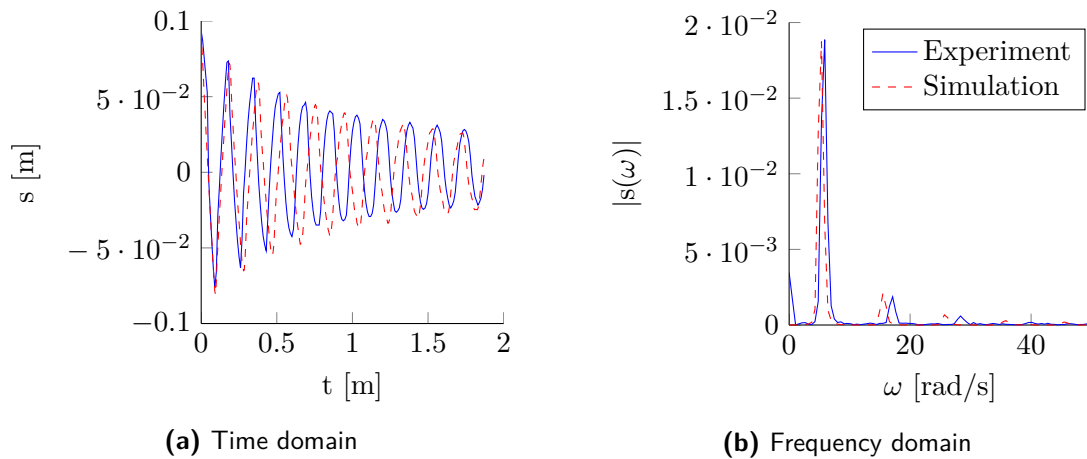
### 8-3-4 Other excitations

Simulating an experiment situation where the string was excited at the  $\frac{1}{4}L$  marker, using the same parameters that resulted in the accurate simulation shown in Figure 8-14, results in a less accurate simulation. Figure 8-16 shows the position of the middle node for simulation and experiment. First, we see that the initial displacements at  $t = 0$  do not match. Possible explanations for this could be that gravity cannot be neglected, or that apart from pulling the string horizontally, it is also moved vertically.

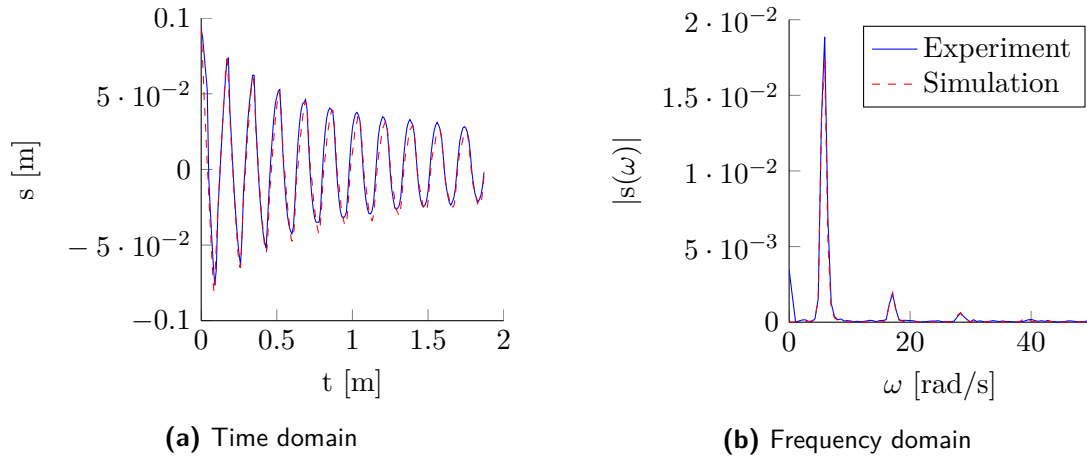
Second, we see that the eigenfrequencies have shifted slightly. This could be explained by the non-linear behavior of the string. This results in the fact that when the magnitude of the excitation is increased, the frequencies of the eigenmodes increase. To illustrate this,



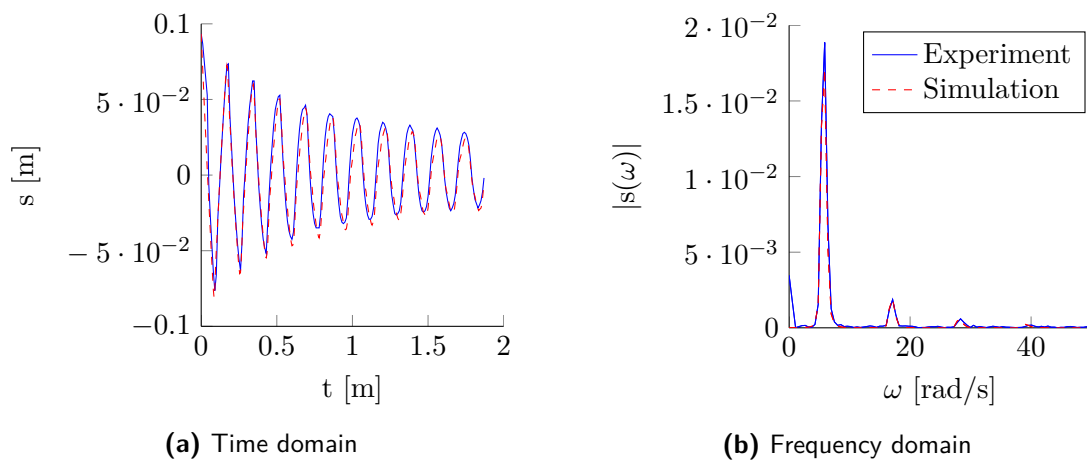
**Figure 8-12:** Time signal and frequency content of middle node displacement for symmetric excitation, experiment versus damped simulation ( $\zeta = 1, C_D = 0$ ).



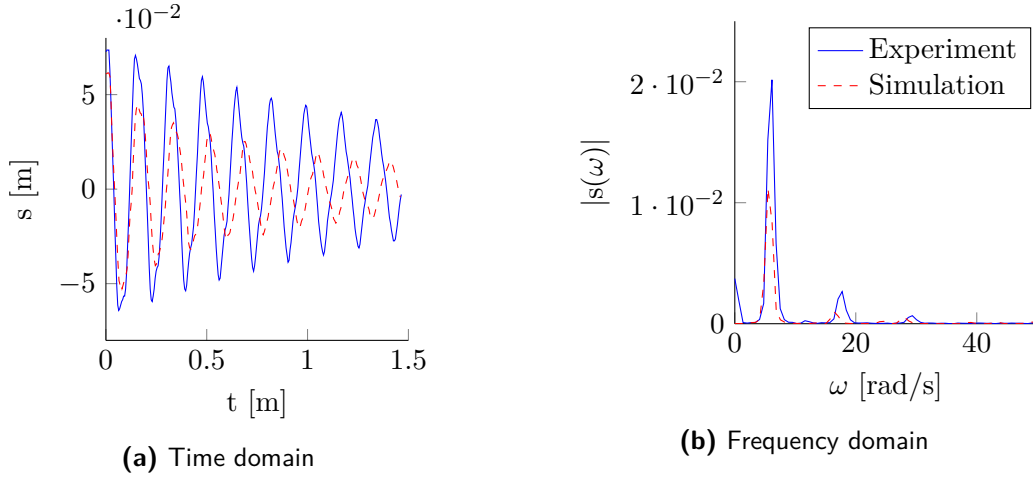
**Figure 8-13:** Time signal and frequency content of middle node displacement for symmetric excitation, experiment versus damped simulation ( $\zeta = 0, C_D = 2$ ).



**Figure 8-14:** Time signal and frequency content of middle node displacement for symmetric excitation, experiment versus damped simulation ( $\zeta = 0, C_D = 2$ ). In this simulation the element stiffness was corrected by a factor of 1.2.



**Figure 8-15:** Time signal and frequency content of middle node displacement for symmetric excitation, experiment versus simulation ( $\zeta = 1, C_D = 2$ ). In this simulation the element stiffness was corrected by a factor of 1.2.



**Figure 8-16:** Time signal and frequency content of middle node displacement for asymmetric excitation, experiment versus simulation ( $\zeta = 0, C_D = 2$ ).

in Figure 8-17 the frequency content of simulations with the same symmetric excitation but with different initial amplitudes are plotted. One can see that the eigenfrequencies of the model increase with the amplitude of the initial excitation, a typical phenomenon in systems in which stiffness is increased due to large deformations.

## 8-4 Stability and convergence

Due to the fixed time-step size and semi-implicit Euler integration method, selecting the right time-step size is crucial for the simulation. It was found for an undamped model of the string that in order to prevent the simulation from becoming unstable, the time-step size needs to satisfy

$$h \lesssim \frac{1}{\omega_e}. \quad (8-21)$$

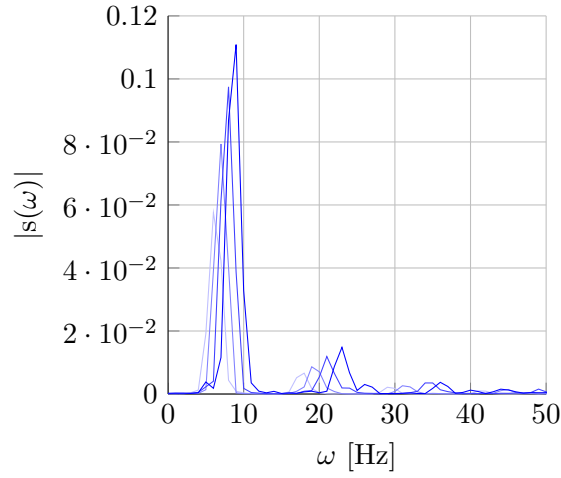
where  $\omega_e$  represents the eigenfrequency of one element of the lumped-mass model, and is defined as

$$\omega_e = \sqrt{\frac{k_e}{m_e}}. \quad (8-22)$$

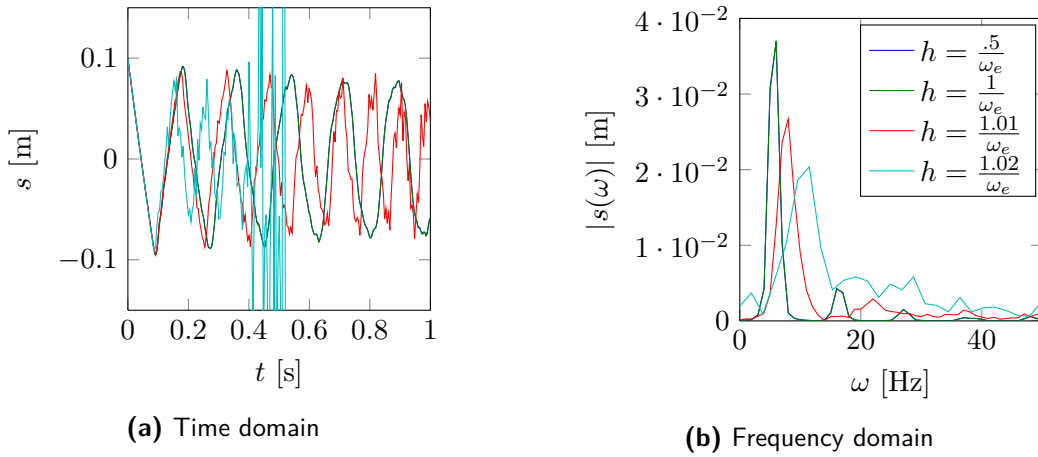
Below this boundary one finds that the relative error scales with roughly  $\mathcal{O}(h^2)$ , as shown in Figure 8-19. However, when

$$h \approx \frac{1}{\omega_e} \quad (8-23)$$

we find that the simulator is not fully unstable, but high frequency noise starts occurring. This is illustrated in Figure 8-18. Also the eigenfrequencies of the model shift upward. Further increasing the time-step size ensures that the system becomes unstable.



**Figure 8-17:** Simulated frequency content of symmetrically excited string for different initial amplitudes. Note the shift in eigenfrequencies.

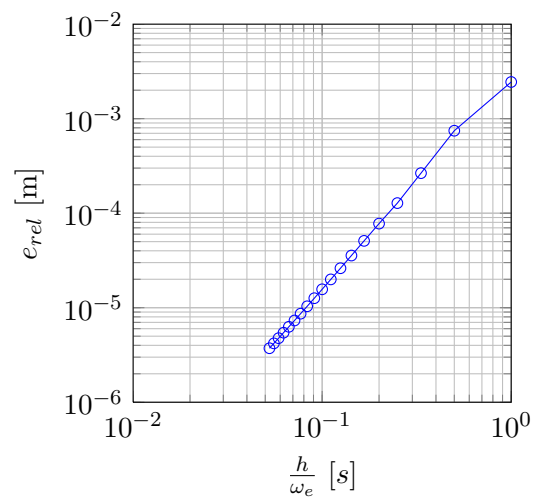


**Figure 8-18:** Middle point displacement in time and frequency domain of simulated string for various time-step sizes around  $h \approx \frac{1}{\omega_e}$ .

One can conclude from this that it is critical to maintain

$$h < \frac{1}{\omega_e} \quad (8-24)$$

and the accuracy achieved at this point is already well enough for the work in this thesis.



**Figure 8-19:** Relative error  $e_{rel}$  versus time-step size relative to  $\frac{1}{\omega_e}$ . The relative error is computed as the maximum absolute difference between two subsequent simulations, each lasting one second.

---

## Chapter 9

---

# Net behavior

Simulation of contact dynamics and elastic systems are brought together in the application of a net wrapping itself around an object. The first part of this chapter describes the test setup and simulation model used to verify net simulations with Bullet & Blender. The second part compares the simulation and experimental results. In the third part a sensitivity analysis of the model is shown.

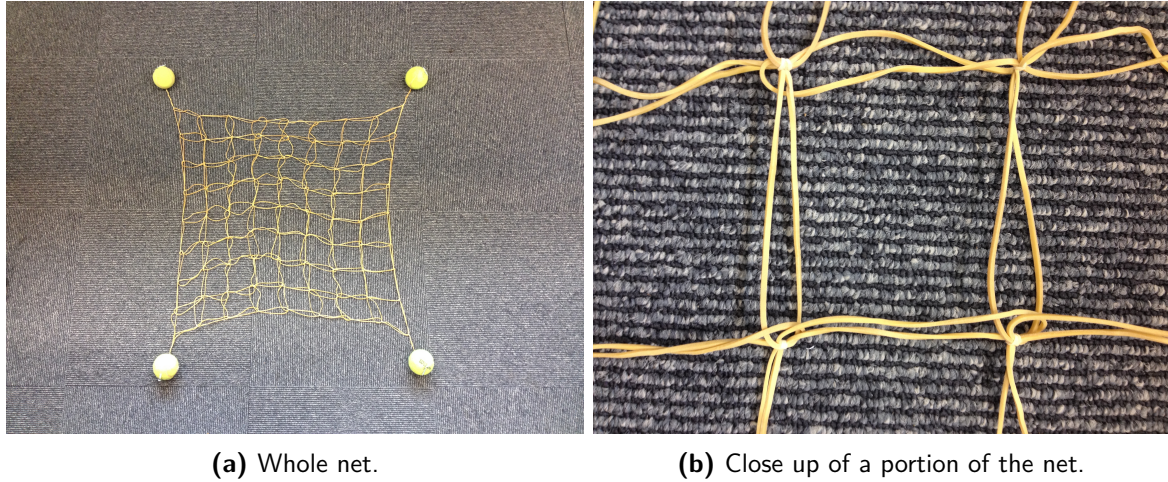
### 9-1 Setup and model

A simple net was constructed by connecting rubber bands in a square matrix form. At the corners of the net tennis-balls were attached as corner-masses, to make the net wrap around the target. The material properties such as weight, natural length and stiffness were measured in a similar way as with the string. The resulting properties are listed in Table 9-1.

Property	Symbol	Value
Band linear density	$\rho_{\text{lin}}$	0.0067 [kg / m]
Band natural length	$l_n$	0.090 [m]
Tennis ball diameter	$d_{tb}$	0.065 [m]
Tennis ball mass	$m_{tb}$	0.055 [kg]

**Table 9-1:** Net properties

At several positions in the net, markers were attached for a Vicon 3D motion capture system. This system, installed in the robotics lab at ESA, consists of 9 infrared cameras and a PC to analyze the images captured by the cameras in real-time. It records the position of the markers attached to the net and export there coordinates in a Cartesian frame. Markers were attached in the center of the net, at the corners of the net, and at the end of the corner masses.



**Figure 9-1:** Net used for experiments. The Vicon 3D motion capture markers have already been removed from the net in these particular photos. The markers attached to the corner masses have been replaced with screws, to keep the rubber bands that go through the balls in position.

The target was constructed out of an inverted dust-bin, a block of wood and a standard pallet, screwed together. Vicon markers were also attached to it to be able to extract the position of the target in the post-processing of the experiments.

The net was dropped over the target by two persons by hand. A third person with a ruler would ensure that the net would be dropped from a similar position in every experiment. Figure 9-2 shows the net and test-setup, prior to being dropped.

The model was implemented in Blender, and is an adaptation of the models used in earlier simulations by ESA to investigate the capturing of satellites[2]. Each rubber band is meshed as four point masses, connected by three sets of linear spring-dampers as used in the string experiments.

The natural length  $l_e$  and mass  $m_e$  of each element are defined as

$$l_e = \frac{l_n}{3}, \quad (9-1)$$

$$m_e = l_e \rho_{\text{lin}}, \quad (9-2)$$

and element stiffness and damping are found in the same way as for the string experiment.

One major difference in the model is that aerodynamic damping was not accounted for in the net simulations, because the model was originally developed for simulating nets in a vacuum.

During initial simulations it was found that the net ‘rolled’ from the target, as shown in Figure 9-3. After investigation the cause was determined to originate from the fact that the masses were allowed to rotate. This caused the nodes to roll in stead of slide across the target. Constraining the angular velocity by setting the type of the bodies to ‘Dynamic’ in Blender prevented this, and yielded the simulation shown in Figure 9-4.





**Figure 9-2:** Net experiment, right before the net was dropped. Note that in this picture the Vicon markers are in fact attached to the net (silver balls).

## 9-2 Experiment versus simulation

Several experiments were performed in which the net was dropped from a similar position. Figure 9-5 shows the measured trajectories and positions of the markers at the end of the masses for 3 experiments. The first thing that can be noted from this is that the spread of the initial position, indicated by the circle shaped markers is in the order of 10 centimeters.

The second thing we note is the fact that the initially the position of the markers follows a similar trajectory in all three experiments, but after reaching their lowest point, they diverge. At this point the system becomes more chaotic, which we will discuss further on in this chapter.

Finally, in Figure 9-5c and Figure 9-5d we can see some high frequency oscillations in the position of the markers. This is most probably due to the fact that the marker is actually hanging behind the ball, and is not positioned in the center of the ball. Therefore rotations of the ball are interpreted by the motion capture system as translational motion.

Figure 9-6 shows a comparison of an experiment and a simulation. Here the measured initial position of the markers is used as the initial position of the corner masses in the simulation.

We can note that the initial drop and wrapping of the net has been modeled relatively accurately. The stiffness seems to be underestimated, resulting in the fact that the corner masses drop lower than in reality. However, the overall movement during the collapse is at least similar to some extent. And as expected we don't see the high frequency oscillations in the simulation, since we are measuring the center of the tennis balls, not the marker at the end.



**Figure 9-3:** Net simulation where the net ‘rolls’ of the target.

However, after the balls have reached the bottom, the behavior is significantly different. This is partly due to the chaotic nature of the system, making it very difficult to simulate exactly.

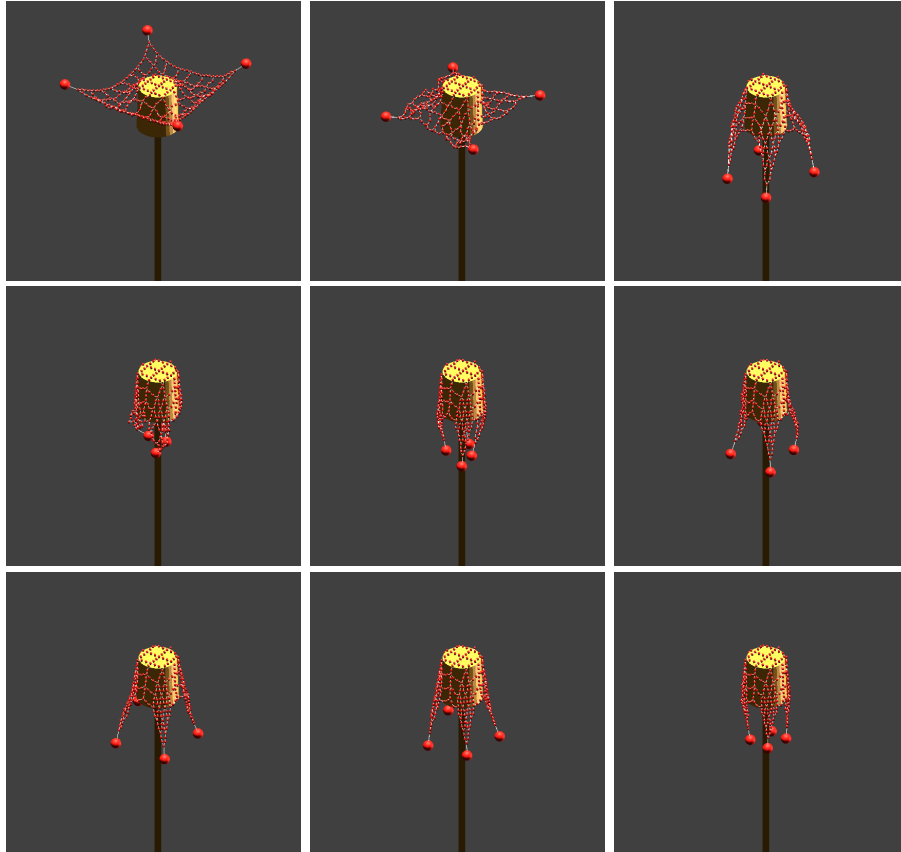
We can also see that in the experiment the balls on the left side are hanging higher than on the right side. This can be explained by the fact that the net had an offset with respect to the center of the target in its initial position. This was not reproduced correctly in the simulation.

Also we see that the net motion seems to damp out faster in simulation than in reality. This could be because the viscous damping coefficient  $c_e$  in the net has been set too high, or due to the fact that the numerical value of the friction coefficient  $\mu$  has been overestimated.

### 9-3 Sensitivity analysis

To show the chaotic nature of the system after the corner masses have reached their lowest point, several sensitivity analyses were performed using Bullet & Blender.

In the first analysis, we compare a simulation where all four masses are released at exactly the same moment, to a simulation where two of the four masses are dropped 50 milliseconds later. The resulting trajectories and positions over time are shown in Figure 9-7. We see that

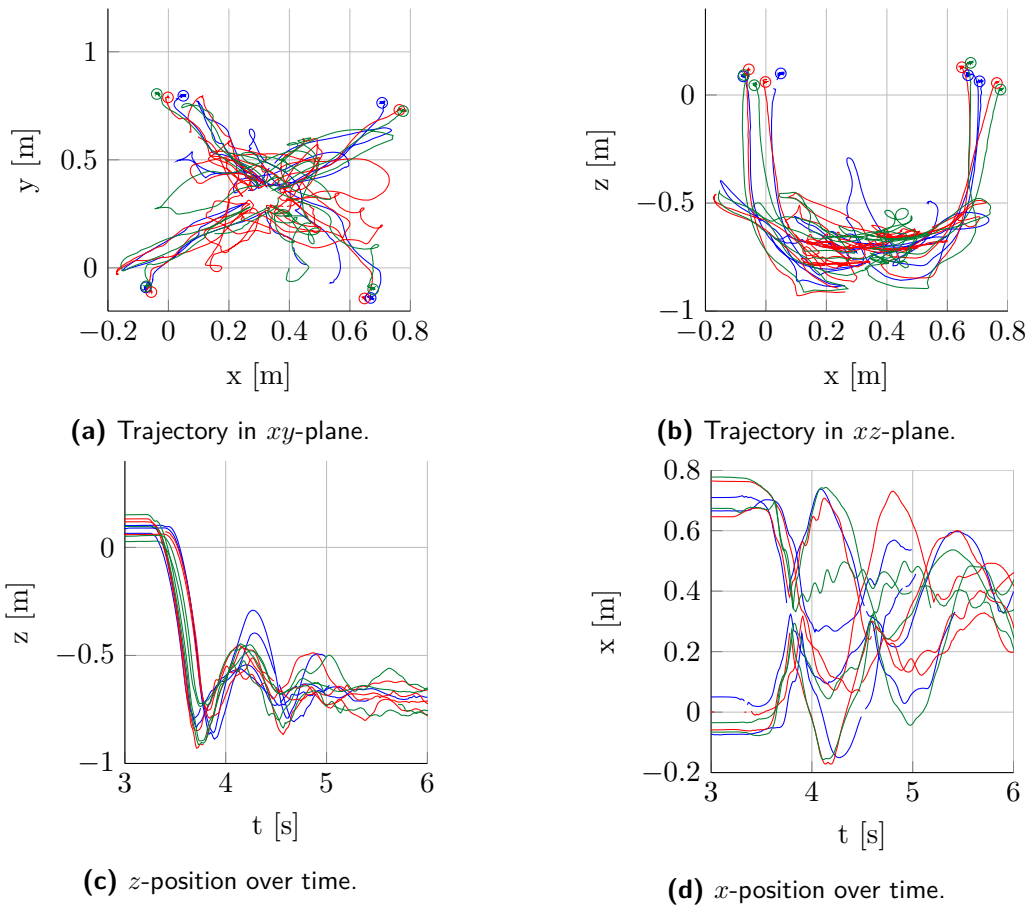


**Figure 9-4:** Net simulation where angular velocities of net masses are constrained, resulting in proper sliding behavior.

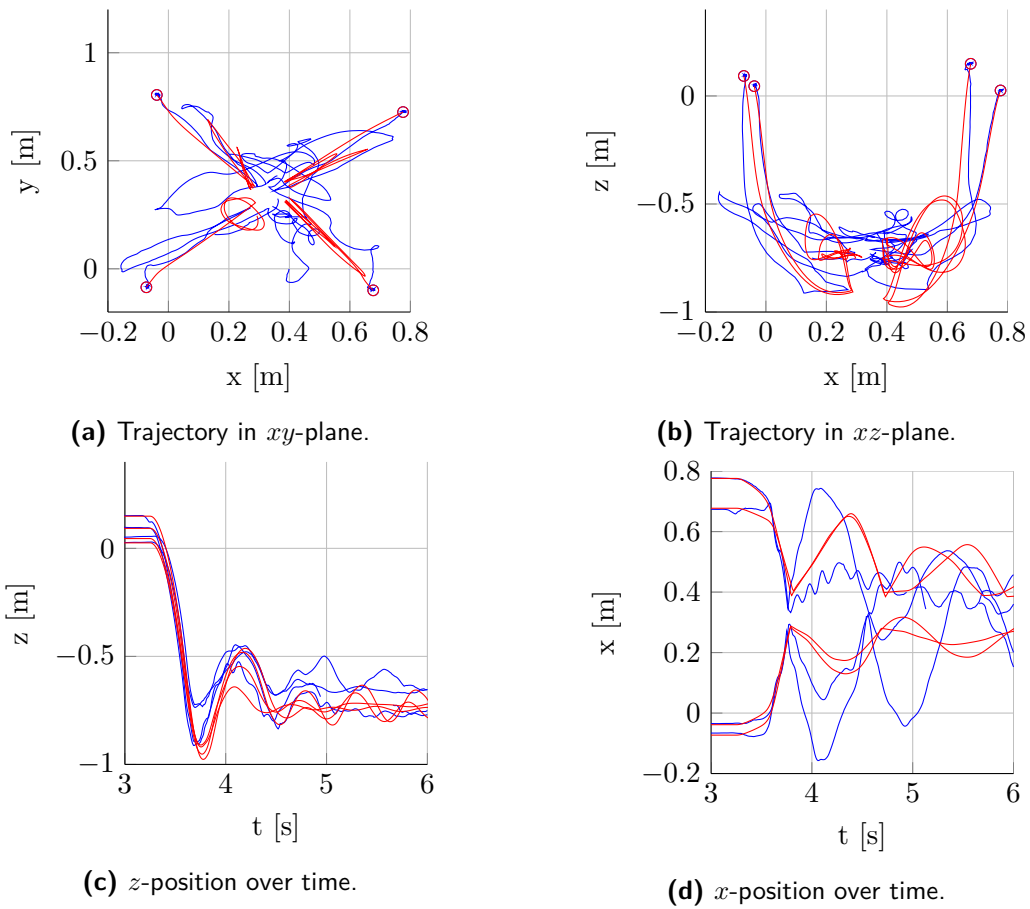
even though the difference in drop time is only small, the resulting trajectories already differ significantly.

In the second analysis, we compare a simulation where all four masses are released from the same vertical ( $z$ ) position, to a simulation where two masses are dropped from a position 50 millimeters higher than the other two masses. The resulting trajectories (Figure 9-8) again shows a big difference between both simulations after the initial wrapping of the net.

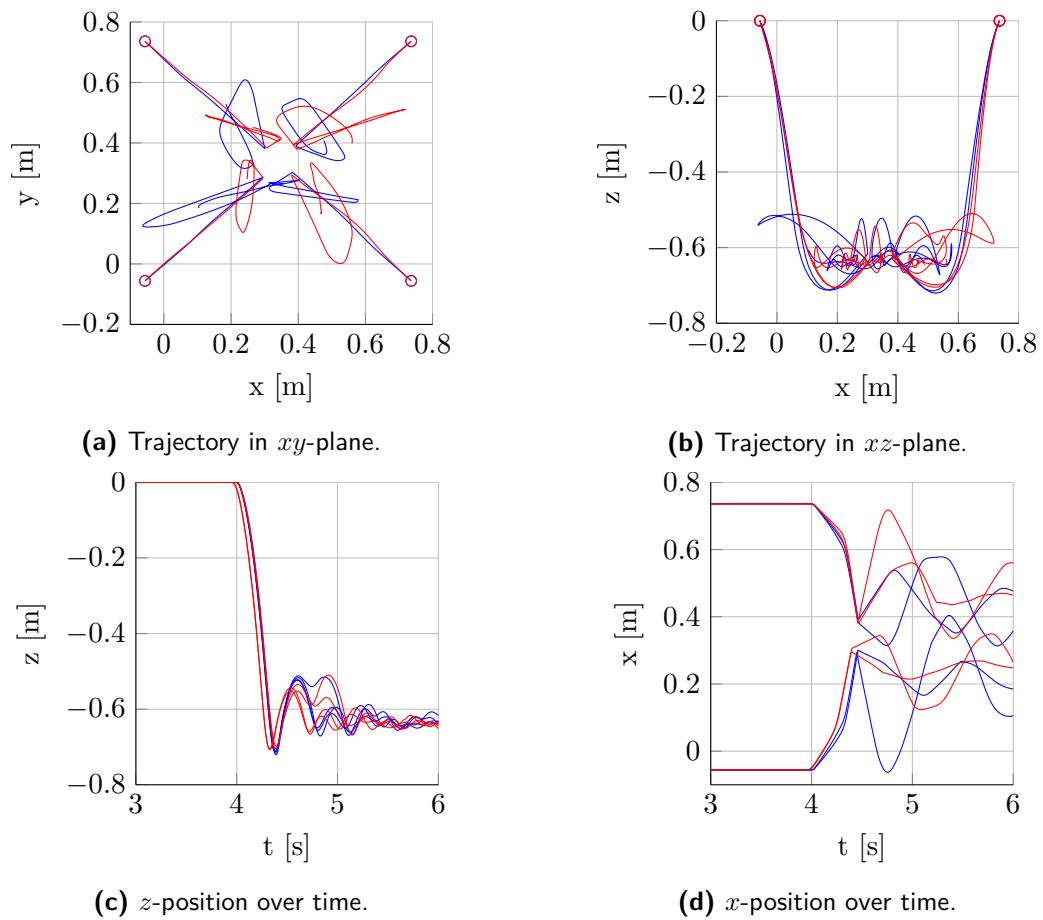
From this we can conclude that, given the rough setup and execution of our current experiments, achieving an exact match between simulation and experiment would be very difficult, if not impossible.



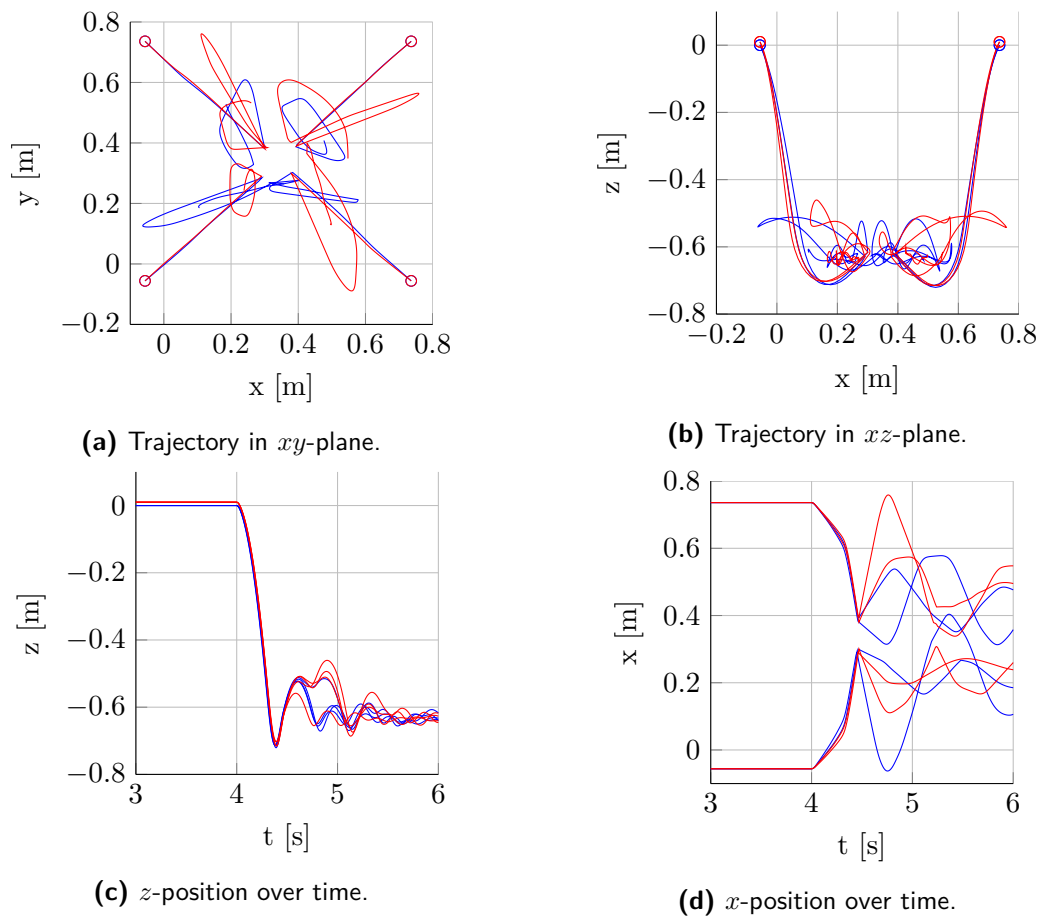
**Figure 9-5:** Measured trajectory and position over time of markers attached to corner masses of net experiment.  $xy$  represents the horizontal plane perpendicular to gravity,  $z$  is the vertical direction.



**Figure 9-6:** Measured position of markers on tennis balls and simulated position of corner masses. Blue represents experimental data, and red the corresponding simulation.



**Figure 9-7:** Sensitivity analysis of simulation of net. In the simulation represented by the blue lines all four corner masses were dropped at the same time, and in the simulation represented by the red lines the two masses were dropped 50 milliseconds later.



**Figure 9-8:** Sensitivity analysis of simulation of net. In the simulation represented by the blue lines all four corner masses were dropped from  $z = 0$  [m], and in the simulation represented by the red lines they are dropped from  $z = 0.05$  [m].





# Computational efficiency

As described in the introduction, earlier research has suggested that gaming engines could be substantially faster in handling certain types of problems. This chapter aims at identifying such a potential for greater computational efficiency of gaming engines by comparing the run time of several models in both Bullet & Blender and MSC/Adams.

The first section discusses the Bullet & Blender's efficiency in handling stiff systems by performing several simulations with the lumped-mass model of the string. The second section focus on contact dynamics, by investigating run time for a simulation where a stack of blocks is collapsed.

### 10-1 Stiff systems

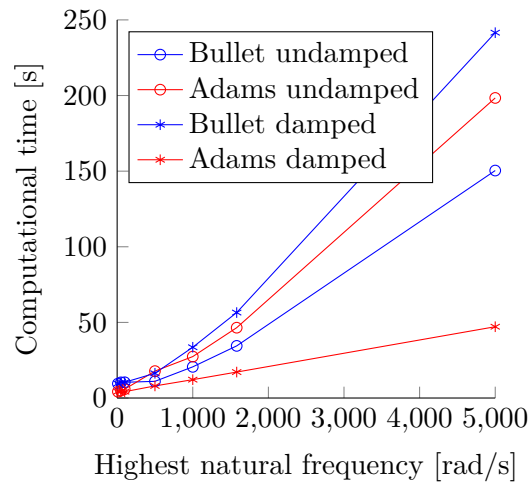
To investigate the computational efficiency of Bullet & Blender and the semi-implicit Euler integration method in handling stiff systems, simulations of the string model were carried out for various levels of damping and stiffness in both Blender and MSC/Adams. As explained before, since scripting models in Adams is beyond the scope of this project, the number of masses used to simulate the string was fixed at 30.

Several values for stiffness and damping were used in simulations with a fixed end time  $t_{\text{end}}$ , and the computation time has been recorded. The fixed time-step size in Bullet & Blender was set such that

$$h = \frac{1}{\omega_e}, \quad (10-1)$$

ensuring a stable simulation with enough accuracy for practical purposes. The results of this analysis are shown in Figure 10-1, where the time required to finish one second of simulation versus element eigenfrequency  $\omega_e$  are plotted for various levels of damping.

It can be seen that for undamped systems, Bullet & Blender and Adams perform comparably. However, once only a little bit of damping is introduced, Adams performance improves, but



**Figure 10-1:** Computational time required to finish a stable simulation with Bullet & Blender, and MSC/Adams.

Bullet's performance deteriorates. One possible explanation for this could be that Adams exploits the fact that in a damped system the high frequency modes die out quickly, thus eliminating the need for a small time-step size. It is plausible that Bullet's performance in this can be improved by adding an adaptive time-stepping algorithm based on some tolerance level.

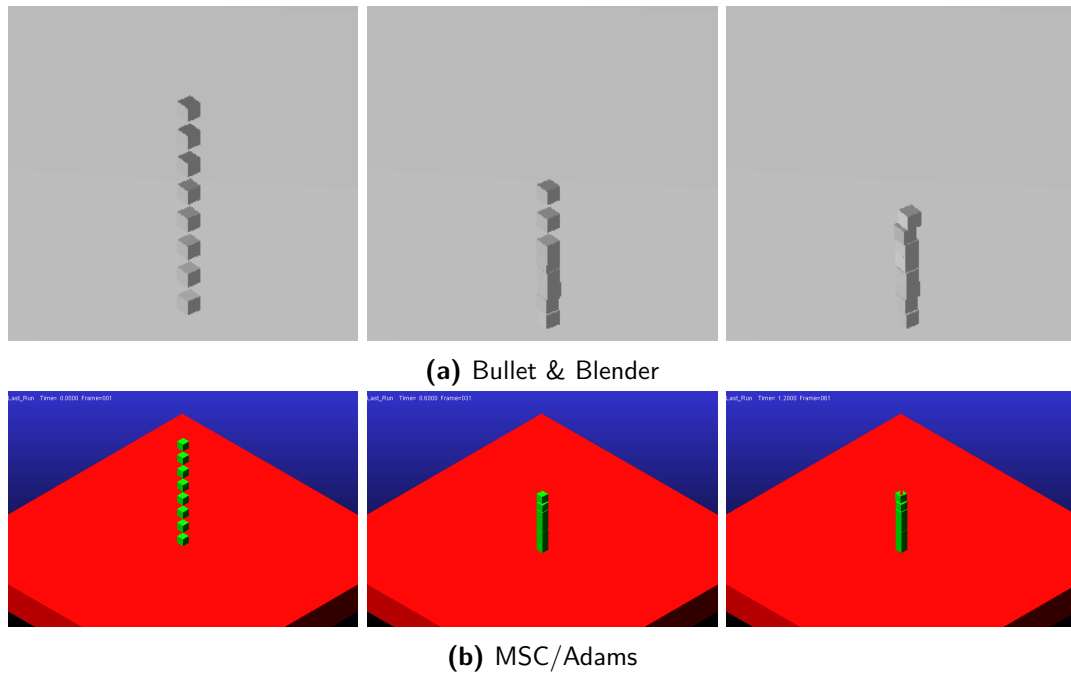
## 10-2 Contact dynamics

To quantify Bullet & Blender's ability in handling contact dynamics a simple collapsing stack of blocks was simulated. There are many things that affect the run time of such a simulation. It was decided to vary the number of blocks, and to vary the friction between objects.

Adams offers the possibility of using a restitution based, or a penalty based model for contact computation. It was decided to use the penalty based method, since this allows us to clearly illustrate the difference between a constraint based, and a penalty based method. The restitution based model employed in Adams is considered to be beyond the scope of this thesis.

Visualization also brings computational expense. Adams has the option to disable real-time rendering during the simulation, but since this is not an option in Blender this was left on. The visualization time-step in both Adams as well as Bullet is set to 0.01 [s]. In Blender the physics sub-step is set to 5, ensuring a fixed physics time-step of 0.002 [s] (see appendix C). Adams defines the physics time-step during the simulation, but allows the user to set a maximum time-step size  $h_{\max}$ . For the experiments this was set to 0.002 [s].

Bullet & Blender offer several types of collision boundaries of bodies for the purpose of computing contact points. In these experiments the boxes are modeled using the 'Box' elementary shapes collision bound. The floor is modeled using a 'Triangle Mesh' collision bound. This greatly affects run time when modeling a large number of blocks. Contact between elementary shapes is much easier to compute than between arbitrary shaped and meshed objects.



**Figure 10-2:** Block simulation for eight blocks with friction coefficient  $\mu = 0.1$ .

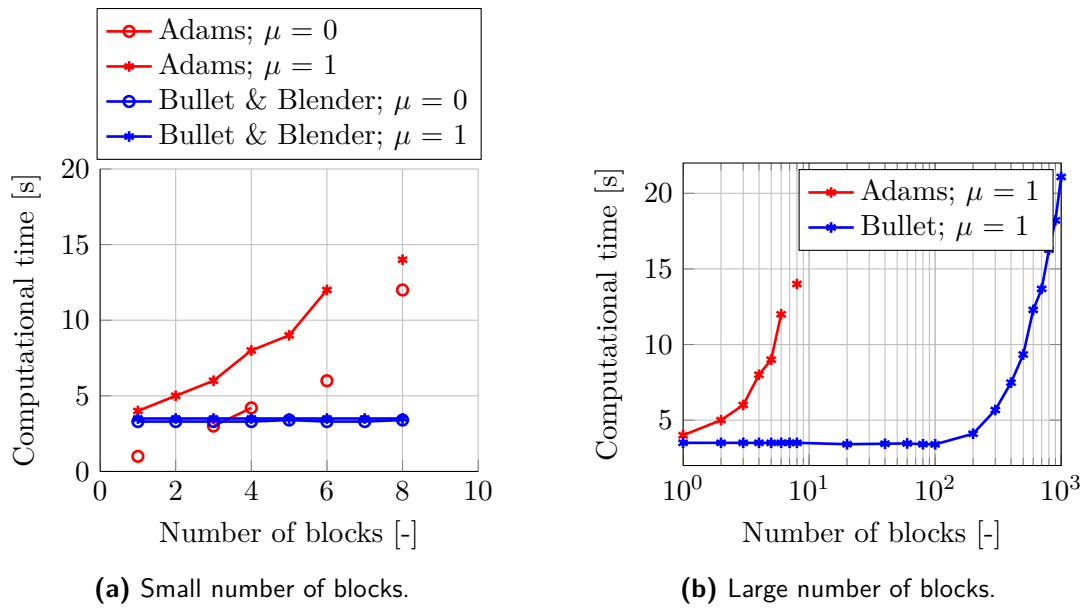
Figure 10-2 shows snapshots from both Blender and Adams from a pair of simulations performed. Note the fact that exact solution to the problem should be eight perfectly aligned boxes stacked on top of each other. However, in both Adams and Blender we see numerical errors arising, that translate themselves to sideways velocities. In this particular example these velocities are damped out, but if no friction were present, the blocks would simply slide apart.

Figure 10-3a shows the run time of three seconds of simulation for Blender and Adams, for two different levels of friction. One can note that the graph representing Adams is sometimes interrupted. In this case Adams was unsuccessful in finishing the simulation. When simulating complex contact, Adams has a much higher tendency to become unstable compared to Bullet.

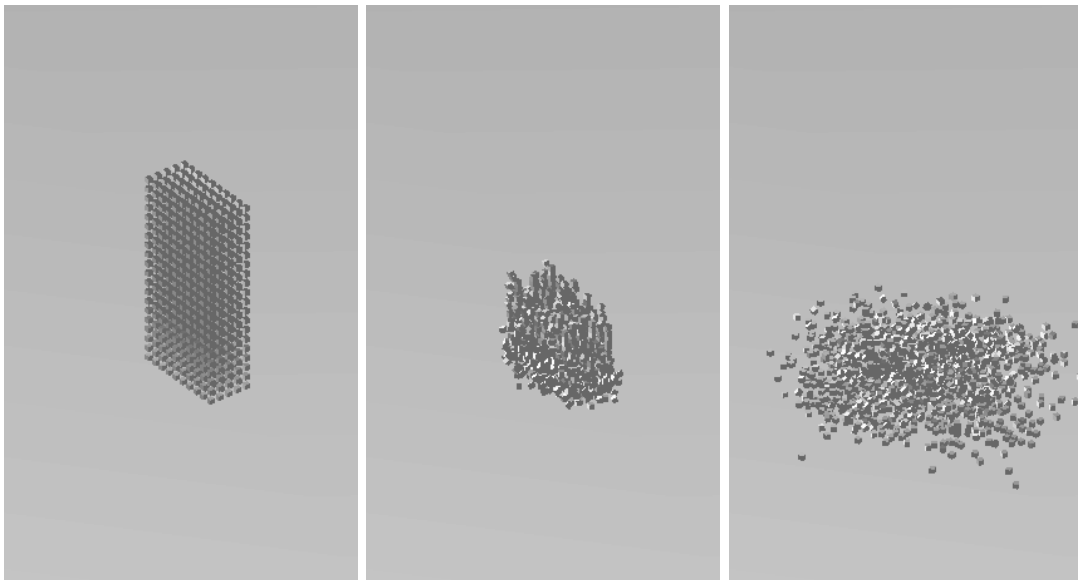
Figure 10-3a also indicates that run-time in Adams increases with the number of blocks, but in Bullet & Blender it stays constant. This is due to the fact that Blender is set to run in real-time, meaning that if the simulation is faster than real-time, the output is merely slowed down. We can conclude that with this number of objects, the physics computation is not the bottleneck in the simulation in Bullet & Blender.

To find the point where physics computation does become the bottleneck in Bullet & Blender, the number of blocks was increased even further. In Figure 10-3b we see that not before we simulate over a hundred bodies does the physics computation become the bottleneck in Bullet & Blender. Screenshots from a simulation of a thousand bodies in Blender is shown in Figure 10-4.

From this analysis we can conclude that Bullet is much better equipped to simulate contact between large numbers of rigid-bodies because of two reasons. First, it is less prone to become unstable. And second, it is several orders of magnitude faster than Adams when it comes to computing contact dynamics.



**Figure 10-3:** Computational time required to perform stable simulation of stacking blocks, for various number of blocks. No entry means that Adams was unable to finish the simulation. Note that real-time simulation lasts three seconds (i.e.  $t_{\text{end}} = 3$  [s]).



**Figure 10-4:** Block simulation for 1000 blocks with friction coefficient  $\mu = 0$  in Blender.

## Conclusions & recommendations

In section 1-2 the research objectives were defined as follows.

- Identify the underlying mathematical methods of gaming engines and assess their accuracy.
- Determine what would be the advantages of using a gaming engines over traditional software.
- Define what applications would be well suited to be simulated with gaming engines.
- Propose a possible implementation for an engineering oriented simulator based on gaming engines.

This chapter will attempt to summarize our findings. The first section will cover the first three questions, and the second section discusses the fourth question.

### 1-1 Conclusions

In chapters 2 and 4 we have shown that the main difference between gaming engines and their engineering counterparts is their focus on real-time applications. This results in the need for simulations with a fixed integration time-step size, which cannot be combined with penalty-based contact models, the latter being a common method for engineering tools to handle rigid-body contact. Instead, gaming engines typically use the Stewart-Trinkle constraint-based method, for handling contacts.

Constraint-based simulation requires either the velocity or the position at the next time-step to be explicit in terms of the forces acting on the bodies. Therefore it is often combined with the semi-implicit Euler integration scheme. This is a symplectic integrator, meaning that it more or less conserves energy. It is shown that the error arising from numerically solving

the Newton-Euler equations using this method, combined with a fixed time-step, scales with  $\mathcal{O}(h)$  for a free falling object. For a simple harmonic oscillator the error in the amplitude and period scale with  $\mathcal{O}(h^2)$ , as long as the time step is smaller than a value related to the natural frequency of the oscillator.

The Stewart-Trinkle method is based on a maximal coordinate formulation. Combined with constraints and numerical integration this formulation generally suffers from stability errors such as joint-drift. This can be solved by adding a constraint stabilization method. Several methods are present, of which Baumgarte is the most simple. However, combining Baumgarte with Stewart-Trinkle results in energy being created in collisions. Post-stabilization seems to be a better alternative, because decreasing the time-step size will in this case yield convergence.

Different levels of friction approximation exist, each with their own deficiencies. When the linearized friction model presented in section 4-4-1 is applied, the exact solution of Coulomb's friction model can be approached by selecting a large number of vectors to span the friction plane. The decoupled friction model presented in section 4-4-2 suffers from overestimation of the static friction force.

The type of friction model that is selected may limit the types of solver that can be used to solve the resulting complementarity problem. Different types of solvers exist, each with their own convergence and computational efficiency characteristics. However, no matter how accurate the friction model and the solution to the complementarity problem, uniqueness is never guaranteed, due to the statistical undetermined nature of systems of rigid-bodies.

Based on all this, we can conclude that depending on the combination of methods that is used, gaming engines can provide accurate solutions to problems involving classical mechanics and rigid-bodies. Generally selecting more accurate methods will yield a more computationally expensive simulation, but this is not true for all cases.

The main advantages of using gaming engines for particular problems are their stability and computational efficiency in handling contact between multiple rigid-bodies. In chapter 10 it was shown that Bullet as a showcase for gaming engines performs several orders of magnitude faster than MSC/Adams when simulating a simple simulation with several rigid-bodies in contact. However, stiff systems and in particular damped systems are not handled with the same ease as is the case in MSC/Adams. A possible explanation for this is that Bullet does not vary the time-step based on the stiffness and harmonic behavior of the system.

In their current form, gaming engines are particularly useful when real-time operation is required. The best known example of this is a simulator. Furthermore gaming engines are very useful when simulating a large number of contact rigid-bodies, or simulations where there is a lot of static contact (i.e. objects lying on each other for a large part of the simulation).

## 1-2 Recommendations

Based on this research we can state that the simulation of space applications can significantly benefit from the use of gaming engines. However, creating a purpose built rigid-body simulator from scratch would require a considerable amount of effort, and the same goes for maintaining it and keeping it up to date.

Bullet & Blender already have a lot of functionality and would be good candidates for adaptation by the engineering community. The reason for this is that they are open-source and therefore flexible and have a strong ongoing development, and hence are constantly being updated with latest advancements in rigid-body simulation.

However, in its current integration in Blender, Bullet shows unphysical behavior in some cases (e.g. friction modeling, elastic collisions). According to Bullet's development team, most of these shortcomings have either already been fixed but are not accessible from Blender, or will be fixed in newer versions of Bullet.

If an engineer were to use Bullet or any other gaming engine, control over the types of algorithms used is preferred. For instance, some simulations might require more accurate friction modeling, which would lead to the linearized friction model and a complementarity problem solver based on the Newton method. However, if the focus is more on speed, and friction accuracy is less stringent, one might prefer using the decoupled friction model with a Projected Gauss-Seidel (PGS) solver. Therefore it is the author's opinion that a tool that allows more control over the gaming engine would be a great asset.

Also additional functionality could be included through extensions of Blender. Blender lacks any post-processing environment. Although everything can be exported to external data files, a more user friendly way to view simulation results would be a great addition.

A module that tracks the total energy of the system could be of use, just as having way to extract reaction forces from the simulation. However, one must always be careful when interpreting these, especially with unilateral constraints. Finally a function to control and read the error in the complementarity problem would be a big improvement.

## 1-3 Further research

One thing that, to the author's best knowledge, has not been investigated much, is the use of adaptive time-stepping for stiff systems in combination with the Stewart-Trinkle method. If this is possible, it might considerably improve the handling of stiff and damped systems by the semi-implicit Euler integration, and could lead to a tool that combines the best of both worlds.

Also, the experimental verification of the net and string should be taken further. More complex, asymmetric excitations of the string were not represented correctly in the simulation. This is perhaps due to the fact that the model does not account for any bending stiffness in the string, which in reality is present. The chaotic nature of the net experiment setup made it very hard to truly verify the accuracy of the simulation method. A better experiment, with a more simple and consistent geometric arrangement would improve the correlation between simulation and experiment.





---

# Appendix A

---

## Tracking

To extract the motion of the tennis ball and the string from the video recording of the experiment, an image tracking tool has been created in Matlab. This has two stages. The first stage is the image processing, in which the video file is adjusted and preprocessed. The second stage extracts the position of the object we want to track. In our case, this second stage uses a method called *correlation*.

### A-1 Preprocessing

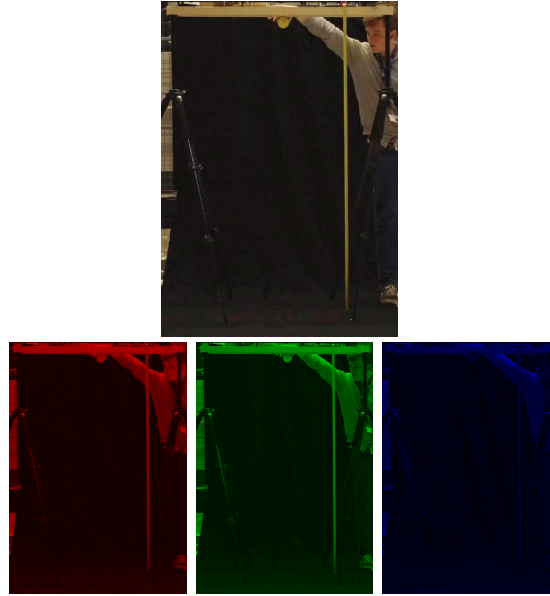
In Matlab a video file is treated as a collection of frames, and each frame is a Red-Green-Blue (RGB) image. This image is represented by a  $N \times M \times 3$  array, where  $N$  and  $M$  represent the dimensions of the image in pixels, and the 3<sup>rd</sup> dimension of the array represent the three color channels. Each entry in this 3D array has a value between 0 and 255 representing a low or high saturation of the respective color channel in that particular pixel. This is illustrated in Figure A-1.

#### A-1-1 Region of interest

First, a region of interest is defined. Since the ball will mainly be bouncing up and down in a small part of the entire frame, the tracker will only need to look for the ball in that area. The tracking algorithm can be sped up significantly by defining this area.

#### A-1-2 Gray-scale & Contrast

The next step is to turn the RGB frame into a black & white, or gray-scale image. This is done through the `rgb2gray(...)` function in Matlab. This effectively turns the frame into a 2D array where each entry has a value between 0 and 255 that correspond to respectively a black or a white pixel or anything in between. Next the contrast of the frame is adjusted to further increase the offset between the bright ball and the dark background. This can be done using the `imadjust(...)` function in Matlab.



**Figure A-1:** Color image and its three color channels.

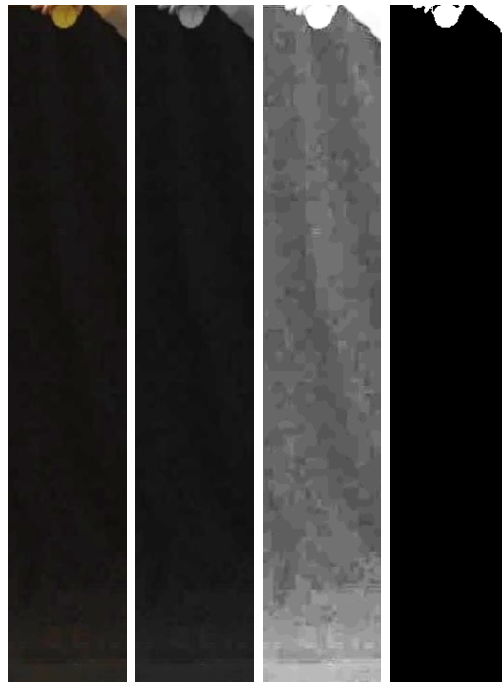
### A-1-3 Apply threshold

The next step is applying a threshold to the image. If the value of a pixel exceeds a certain threshold value, it is set to one, and to zero otherwise. This frame into a 2D array of binaries, or matrix filled with ones and zeros. If the correct values for contrast and threshold are selected, the frame will be black or zero everywhere except the area where the ball is found, since this is lighter than its surroundings.

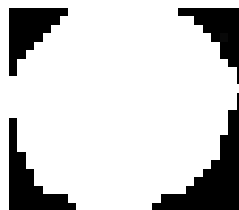
## A-2 Correlation

In this step a *'target'* is selected. This is an exert from an arbitrary binary frame (usually the first) containing the object that needs to be tracked, in our case the ball. This target is again a 2D array containing only ones and zeros, albeit much smaller than the original region of interest. An example of a target is shown in Figure A-3.

We then apply entry-wise multiplication of the target with an area of similar size in the top-left corner of the region of interest. Summing up all these multiplications returns a value, or a score, that represents the match between the target and the chosen area in the region of interest. If the ball was not present in that area, this sum will be zero. However, the larger the similarity between these two regions, the higher the score. This score is saved, and the operation is repeated, but offset one pixel to the right. Once the end of the line is reached, this is repeated one pixel lower. This repeats until this operation was performed at every possible position in the region of interest. Finally, the area with the highest score is designated as the position of the ball.



**Figure A-2:** The three steps applied in preprocessing the video. The first frame depicts the region of interest, the second frame has been turned into a black & white image, the third frame has had its contrast adjusted, and finally the fourth frame is a binary file containing only ones and zeros.



**Figure A-3:** Example of a target that could be used for correlation.



**Figure A-4:** Tracking of one of the experiments. The red asterisk indicates the tracked location of the tennis ball.



---

## Appendix B

---

# Semi-implicit Euler

Numerical integration of the motion of objects in gaming engines is typically performed using a method called the *semi-implicit Euler method* or the *Euler-Cromer method*. This method is a combination of the standard forward (i.e. explicit) and backward (i.e. implicit) Euler integration methods. This appendix explains this method, and the main difference between traditional Euler methods.

Let us define a position of an object, written as  $q$ , with the corresponding velocity  $\dot{q}$ , and acceleration  $\ddot{q}$ . We define the state vector of the system  $\mathbf{y}$  and its derivative as

$$\mathbf{y} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}, \quad \dot{\mathbf{y}} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix}. \quad (\text{B-1})$$

The traditional explicit Euler method takes the form

$$\mathbf{y}^{n+1} = \mathbf{y}^n + h\dot{\mathbf{y}}^n. \quad (\text{B-2})$$

This method lacks stability, and in the case of a harmonic oscillator, the amplitude of the motion will increase over time.

For our system implicit Euler has the form

$$\mathbf{y}^{n+1} = \mathbf{y}^n + h\dot{\mathbf{y}}^{n+1}. \quad (\text{B-3})$$

This method is stable, but in the case of a harmonic oscillator it introduces artificial damping.

The main difference between explicit and implicit Euler is that the state derivate  $\dot{\mathbf{y}}$  is evaluated at the current time-step ( $n$ ) in explicit Euler, and at the next time-step ( $n + 1$ ) in implicit Euler.

It is suggested in literature that semi-implicit Euler originates from explicit Euler, and was discovered by accident [27]. To understand this we split the state space representation of explicit Euler (Eq. (B-2)) in a position update equation, yielding

$$q^{n+1} = q^n + h\dot{q}^n, \quad (\text{B-4a})$$

$$\dot{q}^{n+1} = \dot{q}^n + h\ddot{q}^n. \quad (\text{B-4b})$$

**Algorithm 2** Explicit Euler implementation

---

```

1: procedure EXPLICIT EULER( $q_0, \dot{q}_0, \ddot{q}, t_{\text{end}}$ )
2:   Set  $q = q_0$  and  $\dot{q} = \dot{q}_0$ 
3:   Set  $t = 0$ 
4:   while  $t < t_{\text{end}}$  do
5:      $q = q + h\dot{q}$ 
6:      $\dot{q} = \dot{q} + h\ddot{q}$ 
7:      $t = t + h$ 
8:   end while
9: end procedure

```

---

In a computer this could be written as described in algorithm 2.

Here the values of  $q$  and  $\dot{q}$  at their current time-step are overwritten with their new values. If, by accident, we switch lines 4 and 5, we obtain algorithm 3.

**Algorithm 3** Semi-implicit Euler implementation

---

```

1: procedure SEMI-IMPLICIT EULER( $q_0, \dot{q}_0, \ddot{q}, t_{\text{end}}$ )
2:   Set  $q = q_0$  and  $\dot{q} = \dot{q}_0$ 
3:   Set  $t = 0$ 
4:   while  $t < t_{\text{end}}$  do
5:      $\dot{q} = \dot{q} + h\ddot{q}$ 
6:      $q = q + h\dot{q}$ 
7:      $t = t + h$ 
8:   end while
9: end procedure

```

---

Although this algorithm looks the same, upon closer inspection we see that in this case the position is not updated based on the old velocity. Rather, since the velocity is already overwritten with a new value, it is updated based on the new velocity. This algorithm can therefore be described as

$$\dot{q}^{n+1} = \dot{q}^n + h\ddot{q}^n, \quad (\text{B-5a})$$

$$q^{n+1} = q^n + h\dot{q}^{n+1}. \quad (\text{B-5b})$$

This is the semi-implicit Euler integration scheme. The velocity is updated in similar way as in explicit Euler, and the position is updated as in implicit Euler, hence the name.

---

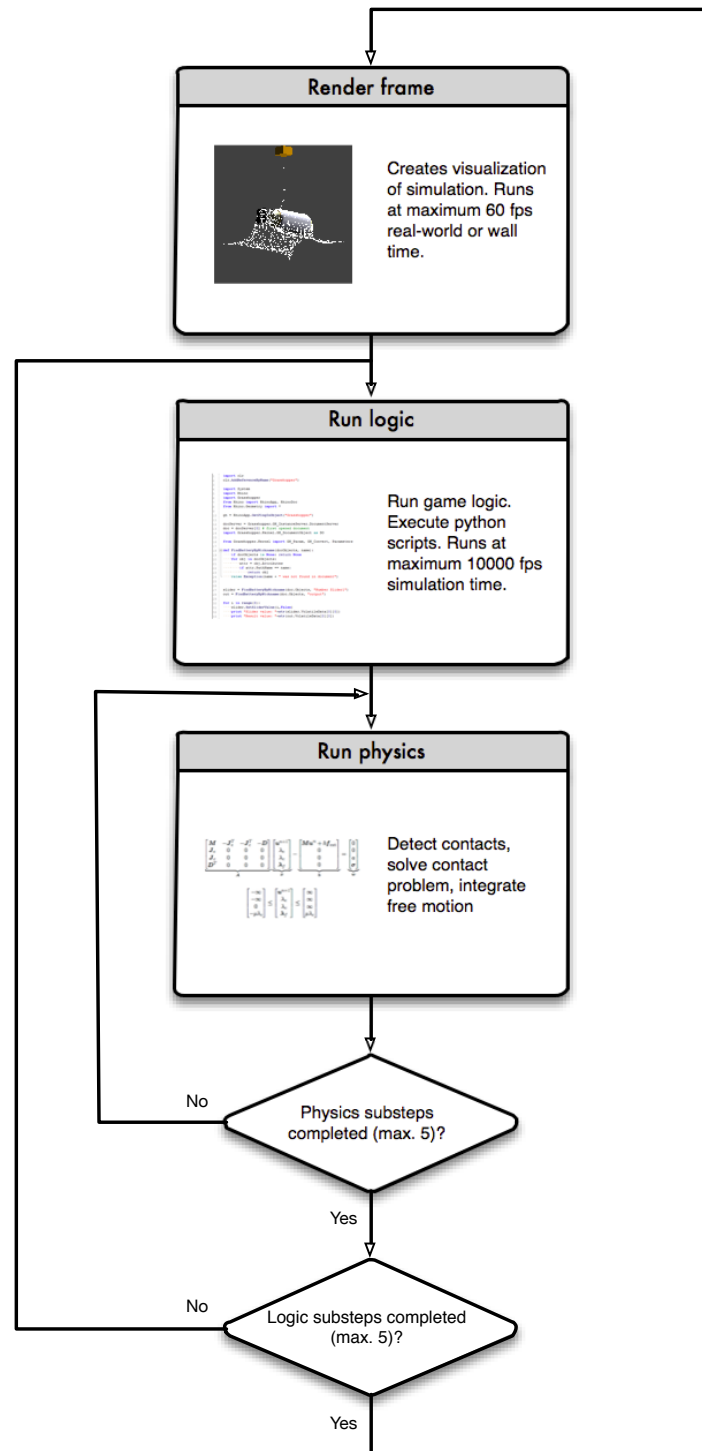
## Appendix C

---

# **Bullet & Blender algorithm structure**

This appendix contains schematic overviews of the identified algorithm structures of Bullet & Blender.

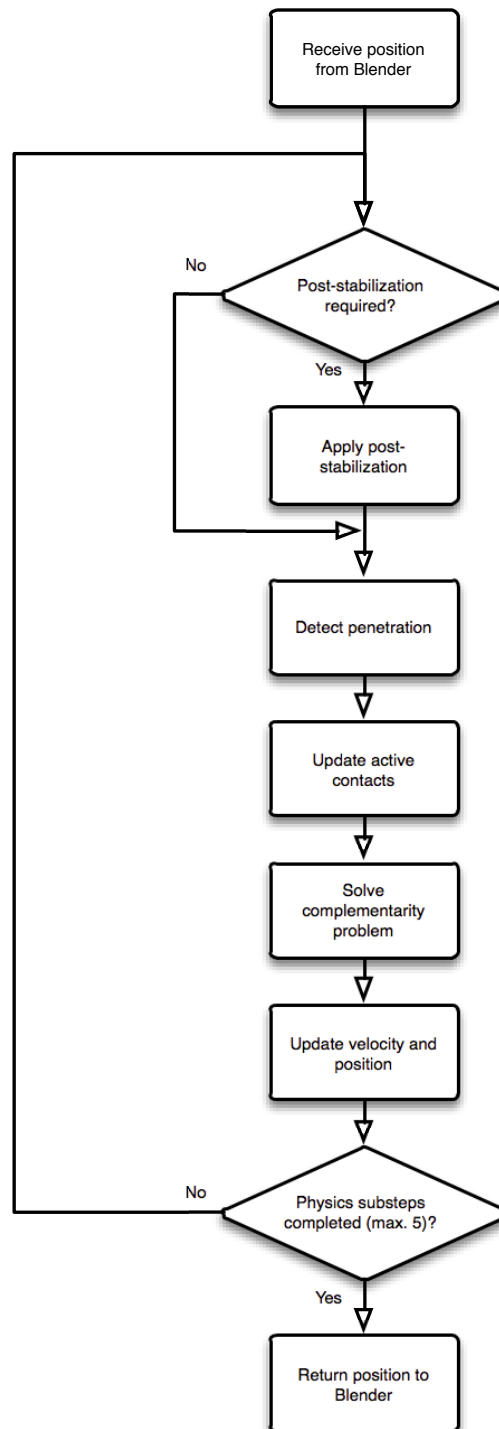
## C-1 Blender



**Figure C-1:** Schematic overview of how Blender handles rendering, scripts and physics.



## C-2 Bullet



**Figure C-2:** Schematic overview of Bullet, as employed by Blender.



---

## List of References

- [1] SIMPACK AG, “Simpack multi-body simulation software.” <http://www.simpack.com>.
- [2] K. Wormnes, J. de Jong, and G. Visentin, “Throw-nets and tethers for robust space debris capture,” in *64<sup>th</sup> International Astronautical Congress, Beijing, China*, 2013.
- [3] M. Software, “Adams.” <http://www.mscsoftware.com/product/adams>.
- [4] E. Coumans, *Bullet 2.80 Physics SDK Manual*, 2012.
- [5] B. Nguyen and Jef, “dvc3d: a three dimensional physical simulation tool for rigid bodies with contacts and coulomb friction,” in *The 1<sup>st</sup> Joint International Conference on Multibody System Dynamics, Lappeenranta, Finland*, 2010.
- [6] A. Price, “Introduction to rigid-body simulations.” <http://www.blenderguru.com/videos/quick-tutorial-make-a-wrecking-ball-with-rigid-body-physics/comment-page-1/>. Accessed on March 2014.
- [7] NVidia, “Popular physics engines comparison.” [www.physxinfo.com](http://www.physxinfo.com), December 2009.
- [8] G. Hippmann, “An algorithm for compliant contact between complexly shaped surfaces in multibody dynamics,” in *Multibody dynamics* (J. A. Ambrósio, ed.), (Lisbon), ID-MEC/IST, July 2003.
- [9] NVidia, “Physx.” [www.physicsinfo.com](http://www.physicsinfo.com).
- [10] Intel, “Havok physics.” <http://www.havok.com/products/physics>.
- [11] R. Smith, “Open dynamics engine.” <http://www.ode.org/>.
- [12] J. Jerez and A. Suero, “Newton game dynamics.” <http://www.newtondynamics.com/>.
- [13] N. Koeng and J. Polo, “Gazebo, 3d multiple robot simulator with dynamics.” <http://gazebosim.org/>.

- [14] Cyberbotics, “Webots 7.” <http://www.cyberbotics.com/>.
- [15] K. Kapellos, “Planetary exploration missions simulation using 3drov,” in *Euromech colloquium on "Nonsmooth contact and impact laws in mechanics"*, Grenoble, France, 2011.
- [16] J. Bender, K. Erleben, J. Trinkle, and E. Coumans, “Interactive simulation of rigid body dynamics in computer graphics,” in *Eurographics* (M. Cani and F. Ganovelli, eds.), (Cagliari, Italy), May 2012.
- [17] P. Lötstedt, “A numerical method for the simulation of mechanical systems with unilateral constraints,” *Department of Numerical Analysis and Computing Science, the Royal Institute of Technology, Stockholm, Sweden*, 1979.
- [18] P. Lötstedt, “Numerical simulation of time-dependent contact and friction problems in rigid body dynamics,” *SIAM Journal of Scientific Statistical Computing*, vol. 5, pp. 370–393, 1984.
- [19] D. Baraff, “Fast contact force computation for nonpenetrating rigid bodies,” *Computer Graphics*, vol. 28, pp. 23–24, 1994.
- [20] D. Stewart and J. Trinkle, “An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction,” *International Journal of Numerical Methods in Engineering*, 1996.
- [21] M. Anitescu and F. Potra, “Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementary problems,” *Nonlinear Dynamics*, vol. 14, pp. 231–247, 1997.
- [22] K. Erleben, *Stable, Robust, and Versatile Multibody Dynamics Animation*. PhD thesis, University of Copenhagen, Denmark, 2005.
- [23] K. Erleben, J. Sporring, K. Henriksen, and H. Dohlmann, *Physics based animation*. Boston, Massachusetts: Cengage Learning, 2005.
- [24] J. Sauer and E. Schömer, “A constraint-based approach to rigid body dynamics for virtual reality applications,” *ACM Symposium on Virtual Reality Software and Technology*, pp. 153–161, 1998.
- [25] J. Baumgarte, “Stabilization of constraints and integrals of motion in dynamical systems,” *Computer Methods in Applied Mechanics and Engineering*, vol. 1, pp. 1–16, 1972.
- [26] M. B. Cline and D. K. Pai, “Post-stabilization for rigid body simulation with contact and constraints,” in *IEEE Intl. Conf. on Robotics & Automation*, vol. 3, (Vancouver, BC), 2003.
- [27] A. Cromer, “Stable solutions using the euler approximation,” *American Journal of Physics*, vol. 49, pp. 455–459, 1981.
- [28] D. Donnely and E. Rogers, “Symplectic integrators: An introduction,” *American Journal of Physics*, vol. 73, pp. 938–945, 2005.

- 
- [29] J. Damkjaer and K. Erleben, "Gpu accelerated tandem traversal of blocked bounding volume hierarchy collision detection for multibody dynamics," in *6<sup>th</sup> Workshop on Virtual Reality Interactions and Physical Simulations*, vol. 1, (Karlsruhe, Germany), pp. 115–124, 2009.
  - [30] J. L. Morales, J. Nocedal, and M. Smelyanskiy, "An algorithm for the fast solution of symmetric linear complementarity problems," *Numerische Mathematik*, vol. 111, pp. 251–266, 2008.
  - [31] M. Silcowitz, S. Niebe, and K. Erleben, "Interactive rigid body dynamics using a projected gauss-seidel subspace minimization method," *Computer Vision, Imaging and Computer Graphics. Theory and applications*, pp. 218–229, 2011.
  - [32] C. G. F. Pfeiffer, *Multibody dynamics with unilateral contacts*. New York: Wiley&Sons, 1996.
  - [33] R. Featherstone, *Robot Dynamics Algorithms*. Kluwer Academic Publishers, Boston/-Dordrecht/Lancaster, 1987.
  - [34] A. Hanson, "Visualizing quaternions," in *SIGGRAPH*, 2005.
  - [35] R. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem*. Boston, Massachusetts: Academic Press, 1992.
  - [36] C. Lemke, "Bimatrix equilibrium points and mathematical programming," *Management Science*, vol. 11, pp. 681–689, 1965.
  - [37] M. Silcowitz, S. Niebe, and K. Erleben, "Nonsmooth newton method for fischer function reformulation of contact force problems for interactive rigid body simulation," *The Visual Computer*, vol. 26, pp. 893–901, 2010.
  - [38] R. van der Linde and A. Schwab, "Multibody dynamics B." Lecture Notes, Delft University of Technology, 2013.
  - [39] O. Mangasarian, "Linear complementarity problems solvable by a single linear program," *Mathematical Programming*, vol. 10, pp. 263–270, 1976.
  - [40] Sony. <http://pro.sony.com/bbsc/ssr/product-NEXFS700UK/>. Accessed on April 2014.
  - [41] F. Alam, S. Watkins, and A. Subic, "The aerodynamic forces on a series of tennis balls," in *15<sup>th</sup> Australasian Fluid Mechanics Conference, The University of Sydney, Sydney, Australia*, December 2004.



---

# Glossary

## List of Acronyms

<b>AMD</b>	Advanced Micro Devices
<b>LCP</b>	Linear Complementarity Problem
<b>GPU</b>	Graphics Processing Unit
<b>MLCP</b>	Mixed Linear Complementarity Problem
<b>MNCP</b>	Mixed Non-linear Complementarity Problem
<b>PGS</b>	Projected Gauss-Seidel
<b>COM</b>	Centre Of Mass
<b>FPS</b>	Frames Per Second
<b>BGE</b>	Blender Game Engine
<b>RGB</b>	Red-Green-Blue

## List of Symbols

$\alpha$	Baumgarte stabilization parameter
$\omega$	Rotational velocity vector
$\sigma$	Slack variable with no physical meaning
$\delta_{ij} \bullet$	Entry-wise subtraction of two vectors related to masses $i$ and $j$
$\epsilon$	Coefficient of restitution or strain
$\eta$	Number of vectors used to approximate the the friction cone
$\gamma$	Slack variable that can be interpreted as approximation of sliding velocity
$\lambda_c$	Lagrange multiplier for inequality constraints

$\lambda_e$	Lagrange multiplier for equality constraints
$\lambda_{f,i}$	Lagrange multiplier for friction impulse
$\mu$	Friction coefficient
$\omega$	Natural frequency of harmonic oscillator
$\omega_e$	Element natural frequency
$\phi(\mathbf{s})$	Bilateral/equality constraint
$\psi(\mathbf{s})$	Unilateral constraint
$\rho$	Radius
$\rho_{\text{air}}$	Density of air
$\rho_{\text{lin}}$	Cord linear density
$\sigma$	Normal stress
$\zeta$	Critical damping ratio
$\zeta$	Slack variable with no physical meaning
$\mathbf{A}$	Complementarity problem matrix
$\mathbf{b}$	Complementarity problem vector
$\mathbf{D}$	Friction cone linearization matrix or diagonal matrix
$\mathbf{E}$	All-ones matrix used for friction model
$\mathbf{f}_{\text{contact}}(\mathbf{s}, \mathbf{u})$	Contact force vector
$\mathbf{f}_{\text{ext}}(t, \mathbf{s}, \mathbf{u})$	External force vector
$\mathbf{F}_{d,ij}$	Spring force between masses $i$ and $j$
$\mathbf{F}_{D,i}$	Aerodynamic drag force acting on mass $i$
$\mathbf{F}_{s,ij}$	Spring force between masses $i$ and $j$
$\mathbf{G}$	Constraint Jacobian without kinematic map
$\mathbf{I}$	Inertia tensor
$\mathbf{J}_c$	Jacobian of inequality constraint
$\mathbf{J}_e$	Jacobian of equality constraints
$\mathbf{L}$	Strictly lower diagonal matrix
$\mathbf{M}$	Generalized mass matrix
$\mathbf{p}_c$	Contact impulse
$\mathbf{p}_e$	Equality constraint impulse
$\mathbf{p}_f$	Friction impulse
$\mathbf{q}$	Unit quaternion
$\mathbf{r}$	Position vector
$\mathbf{s}$	Generalized position vector
$\mathbf{S}(\mathbf{s})$	Kinematic map
$\mathbf{U}$	Strictly upper diagonal matrix
$\mathbf{u}$	Generalized velocity vector
$\mathbf{v}$	Translational velocity vector
$\mathbf{w}$	Complementarity problem residue vector
$\mathbf{x}$	Complementarity problem solution vector



---

$\mathbf{x}_l$	Complementarity problem lower boundary
$\mathbf{x}_u$	Complementarity problem upper boundary
$\mathcal{T}$	Kinetic energy
$\mathcal{V}$	Potential energy
$\mathbf{a}$	Relative velocity orthogonal to contact plane
$A_h$	Cord cross-sectional area perpendicular to horizontal direction
$\mathbf{b}_B$	Complementarity problem vector term related to Baumgarte stabilization
$\mathbf{b}_e$	Complementarity problem vector term related to elastic collisions
$C_D$	Drag coefficient
$c_e$	Element damping coefficient
$d$	Cord diameter
$d_{tb}$	Tennis ball diameter
$E$	Young's modulus
$g$	Gravitational acceleration
$h$	Time-step size
$k$	Number of contact points
$k_{\text{erp}}$	Error reduction parameter
$k_e$	Element stiffness
$l_e$	Element natural length
$l_n$	Cord natural length
$l_t$	Cord length under pretension
$m$	Number of equality constraints
$m_e$	Element mass
$m_{tb}$	Tennis ball mass
$N$	Number of mass elements
$\bullet'$	Denotes parameters related to the probing experiment
$\hat{\bullet}$	Denotes unit vector

