Universität Heidelberg
Fakultät für Mathematik und Informatik
Universitätsmedizin Mannheim

Fortgeschrittenen Praktikum

Improving Guidewire Simulations in Unity

Name: Maximilian Maria Richter
Matrikelnummer: 3463789
Betreuer: Prof. Dr. Jürgen Hesser
Datum der Abgabe: February 24, 2026

Declaration of Authorship

I hereby certify that I have written the work myself and that I have not used any sources or aids other than those specified and that I have marked what has been taken over from other people's works, either verbatim or in terms of content, as foreign. I also certify that the electronic version of my thesis transmitted completely corresponds in content and wording to the printed version. I agree that this electronic version is being checked for plagiarism at the university using plagiarism software.

first and last name	city, date and signature

Abstract

The simulation of medical guidewires has been a challenging problem in the field of computational medicine. The guidewire is a thin and flexible rod that is used in medical procedures to navigate through the human body. The simulation of guidewire is important for training medical students and for planning medical procedures. In this thesis, we present a improved method for simulating guidewire using the Cosserat rod theory in the Unity game engine. The Cosserat rod theory is a mathematical model that describes the behavior of thin and flexible rods. We use the Cosserat rod theory to model the guidewire as a series of connected particles. We then simulate the guidewire using the position and orientation-based dynamics. Instead of using the standard Euler integration, we implement the second-order symplectic verlet scheme. Additionally, we enabled large rod-element lengths by implementing collisions with these rod elements. These changes had a significnat effect on the stability and computation efficiency, compared to previous approaches. We validate our method by comparing the simulation results with the analytical solution of the Cosserat rod theory. For more complicated cases, we created a ground-truth and compared the solution for different parameters to this reference. Our method is able to simulate the guidewire with high accuracy and efficiency. We believe that our method can be used in medical training and planning applications.

ZUSAMMENFASSUNG

Die Simulation von medizinischen Führungsdrähten stellt ein anspruchsvolles Problem im Bereich der computergestützten Medizin dar. Der Führungsdraht ist ein dünner und flexibler Stab, der in medizinischen Verfahren verwendet wird, um sich durch den menschlichen Körper zu bewegen. Die Simulation des Führungsdrahtes ist wichtig für die Ausbildung von Medizinstudenten und für die Planung medizinischer Eingriffe. In dieser Arbeit präsentieren wir eine verbesserte Methode zur Simulation von Führungsdrähten unter Verwendung der Cosserat-Stab-Theorie in der Unity-Game-Engine. Die Cosserat-Stab-Theorie ist ein mathematisches Modell, das das Verhalten von dünnen und flexiblen Stäben beschreibt. Wir verwenden die Cosserat-Stab-Theorie, um den Führungsdraht als eine Serie von verbundenen Partikeln zu modellieren. Anschließend simulieren wir den Führungsdraht mittels positions- und orientierungsbasierter Dynamik. Anstelle der Standard-Euler-Integration implementieren wir das symplektische Verlet-Schema zweiter Ordnung. Darüber hinaus ermöglichen wir durch die Implementierung von Kollisionen mit diesen Stabelementen große Stabelementlängen. Diese Änderungen hatten einen signifikanten Einfluss auf die Stabilität und Recheneffizienz im Vergleich zu früheren Ansätzen. Wir validieren unsere Methode, indem wir die Simulationsergebnisse mit der analytischen Lösung der Cosserat-Stab-Theorie vergleichen. Für komplexere Fälle erstellten wir eine Ground-Truth und verglichen die Lösung für verschiedene Parameter mit dieser Referenz. Unsere Methode ist in der Lage, den Führungsdraht mit hoher Genauigkeit und Effizienz zu simulieren. Wir glauben, dass unsere Methode in der medizinischen Ausbildung und Planungsanwendungen eingesetzt werden kann.

Contents

1 Introduction

As computational power continues to advance, real-time simulation of physical systems with high accuracy is becoming increasingly feasible. This progress is particularly significant in the field of medicine, where it offers transformative opportunities for training. For instance, young physicians can now learn and refine new medical intervention techniques using computer simulators, reducing the reliance on live patients for their training.

One critical area where these simulations have a profound impact is in cardiovascular surgery, which presents unique challenges due to the complexity of the human heart and blood vessels. Traditional open-heart surgeries are increasingly being replaced by less invasive procedures, such as those involving guidewires to insert stents. Guidewires, which are thin, flexible, yet sturdy rods, are threaded through the aorta–the body's main artery, responsible for carrying oxygen-rich blood from the heart to the rest of the body. The precise navigation of these guidewires through blood vessels is a delicate task that demands extensive practice.

This is where computer simulations become invaluable. Accurately simulating the behavior of guidewires within the human vascular system requires models that are both physically accurate and computationally efficient. Research by Viellieber [Vie23] has demonstrated that the Cosserat rod model, introduced by Kugelstadt and Schömer [KS16], can effectively simulate guidewires. This model builds upon the Position-Based Dynamics (PBD) framework developed by Müller et al. [Mül+07], originally designed for simulating soft bodies in computer games.

In this work, we aim to build upon this foundation and further enhance the simulation of guidewires using the Cosserat rod model. By improving the accuracy, stability, and efficiency of the simulation, we hope to contribute to the development of more effective training tools for medical professionals. Our work will focus on refining the existing simulation code, optimizing its performance, and exploring new techniques to enhance the realism and usability of

the simulation. In the following chapters 2 and 3, we will delve into the theoretical underpinnings of our work and outline the methods we will employ to achieve our objectives. We will then present our results and discuss their implications in chapter 4, followed by a discussion in chapter 5 and conclusion in chapter 6.

2 Theory

The following chapter is dedicated to the fundamentals of this work. First, we introduce position-based dynamics and its application to the cosserat rod model to simulate guidewires for medical intervention. We briefly discuss the implementation of the simulation in the Unity game engine and end this chapter by looking into the theory of time integration in physical simulations in search for potential improvements.

2.1 Guidewire Simulation

A guidewire can be conceptualized as a soft body, consisting of a series of points linked by rod elements along a single-dimensional path. Each point is typically connected to two rods, except at the guidewire's ends, where only one rod is present. Simulating this system can be approached using the Newtonian/Lagrangian framework, which involves solving the equations of motion for each point while enforcing constraints. However, this requires dealing with a complex, nonlinear system of equations. Alternatively, the position-based dynamics approach provides a different and potentially more straightforward method for simulating such soft bodies, as outlined below.

2.1.1 Position Based Dynamics

The idea to invent a new approximate framework to simulate soft bodies was fueled by the game industry. Here, it was more important to have "real looking" simulations than physically correct ones. Amongst the most popular methods to simulate such soft bodies are the approaches based on the PBD by Müller et al. [Mül+07], where overshooting problems and explicit integration schemes can be avoided. PBD is also used in the PhysX physics engine by NVIDIA.

In PDB, the first step is the prediction of the particle movement based on their current velocities, usually using the simple Euler integration (See Section 2.2.1).

In the second step, all collisions with external objects are registered. In the third step, the constraints are solved iteratively, e.g., distance or collision constraints. Each constraint requires a unique Lagrange multiplier, such that with each iteration, the positions are corrected slowly in the direction the constraints are satisfied. Let C be the constraint function, which is zero if the constraint is satisfied, i.e., $C(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) = 0$. A correction $\Delta\mathbf{p}$ is then calculated by solving the following approximate equation

$$C(\mathbf{p} + \Delta\mathbf{p}) \approx C(\mathbf{p}) + \nabla_{\mathbf{p}} C(\mathbf{p}) \Delta\mathbf{p} = 0. \tag{2.1}$$

To restrict $\Delta\mathbf{p}$ to the constraint manifold, the Lagrange multiplier λ is introduced, such that the correction is given by

$$\Delta\mathbf{p} = -\lambda \nabla_{\mathbf{p}} C(\mathbf{p}). \tag{2.2}$$

Finally, using Equation 2.1 and Equation 2.2, the formula for λ can be derived as

$$\lambda = \frac{C(\mathbf{p})}{\sum_i |\nabla_{\mathbf{p}_i} C(\mathbf{p})|^2 / m_i}, \tag{2.3}$$

where m_i is the mass of the i-th particle.

After the constraint solving step, the new velocities are calculated based on the old and new positions $\mathbf{p}_i$ and $\mathbf{x}_i$, respectively, i.e,

$$\mathbf{v}_i = \frac{\mathbf{p}_i - \mathbf{x}_i}{\Delta t}. \tag{2.4}$$

The new velocities are then used to predict the next positions in the next time step.

2.1.2 Corresat Rod Model

Cosserat rods are the generalization of Kirchhoff rods, which are used to simulate elastic rods, using not only the positions of the particles and bending between rod elements, but also the orientation or twisting of the rods. This enables the simulation of all possible modes possible in such a system.

Kugelstadt and Schömer [KS16] showed that the cosserat rod model can be formulated in the PBD framework by constraining coupled orientations, using only quaternion multiplication and addition. This approach is very stable and fast, while providing reasonable physical accuracy for computer games.

2.1.3 Simulation Loop

The guidewire simulation used in this work was initially developed by Viellieber [Vie23] in his master's thesis and implemented in the Unity game engine [Uni23]. The code was further improved by Kreibich [Kre23] in his bachelor's thesis and first convergence studies have been conducted. The work of this practical is aimed to continue the improvements of the code.

Unity offers a programming API in C#, which is a strongly-typed, object-oriented programming language. Therefore, all the components of the simulation have to formulated as classes with corresponding member variables and methods. This ensures a consistent interface and easy extensibility of the code.

The simulation loop consists of the following steps:

1. **Prediction**: The positions of the particles are predicted based on their current velocities using the implicit Euler integration.

2. **Collision Detection**: The collision detection is performed using the GJK algorithm. If a collision is detected, the penetration depth and normal are calculated.

3. **Constraint Solver**: The constraints are solved iteratively. The constraints are the distance constraints between the particles and the bending constraints between the rod elements. Each constraint requires a unique Lagrange multiplier, such that with each iteration, the positions are corrected slowly in the direction the constraints are satisfied.

4. **Update Velocities**: The new velocities are calculated based on the old and new positions according to Equation 2.4.

2.2 Time Integration

Time integration plays a pivotal role in the success of any physical simulation. To numerically solve the ordinary differential equations (ODEs) derived from Newton's equations of motion, time must be discretized into discrete steps, denoted as $\Delta t = t_{k+1} - t_k$. Time integration techniques are then employed to update the system's state at each of these time steps. Various methods are available for this purpose, each with its own trade-offs in terms of accuracy, stability, and computational efficiency. In the following section, we will explore the most commonly used time integration methods in physical simulations, examining their strengths and applications to help you choose the best approach for your needs.

2.2.1 Explicit Euler

The explicit Euler integration estimates the derivative of the ODE by Taylor expanding the equation up to first order. Let $\mathbf{x}_k$ be the position and $\mathbf{v}_k = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{x}_k$ the velocity of the system at time t_k. The update step to get from t_k to t_{k+1} is then simply given by

$$\mathbf{v}_{k+1} = \mathbf{v}_k + \Delta t \mathbf{a}_k, \tag{2.5}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta t \mathbf{v}_k, \tag{2.6}$$

where $\mathbf{a}_k = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}_k = \sum_i \frac{\mathbf{F}_{i,\mathrm{ext}}}{m_i}$ is the acceleration of the system at time t_k, which is usually provided by external forces $\mathbf{F}_{i,\mathrm{ext}}$, e.g., gravity, and weighted by the inverse mass of the i-th particle $1/m_i$ according to Newtons law. This method is very simple and easy to implement, but it is not very accurate. The error of the Euler method is proportional to the step size Δt.

2.2.2 Verlet Scheme

The Verlet or Strömer integration [Ken14] is a symplectic integrator, which means that it conserves the energy of the system. The update step is

$$\mathbf{x}_{k+1} = 2\mathbf{x}_k - \mathbf{x}_{k-1} + \Delta t^2 \mathbf{a}_k. \tag{2.7}$$

Verlet integration, while more accurate than the Euler method, is not self-starting, i.e., it requires two initial states to start a simulation. The error associated with Verlet is proportional to the square of the time step, Δt^2, making it a second-order method. Its robustness and energy-conserving properties have made it a standard in molecular dynamics simulations. However, the separate computation of velocities can be resource-intensive, and the method requires additional memory to store the previous state. In guidewire simulations, Verlet integration is useful for updating particle positions, with velocities calculated separately as outlined in Equation 2.4.

3 Method

In this chapter, we present the improvements made to the simulation, such as the variable rod element length, the collision with cylinders, and the control of the guidewire. We end this chapter by discussing the improvements made to the software quality of the simulation.

3.1 Simulation Improvements

One of the main objectives of this work is to improve the simulation speed while keeping the accuracy of the computations in physical meaningful bounds. To achieve this, we implemented several improvements to the simulation of guidewires in Unity.

3.1.1 Variable Rod Element Length

The most simple way to achieve a faster simulation is to reduce the number of spheres in the guidewire, or conversely, the number of rod elements. To sustain the same length of the guidewire, the length of the rod elements has to be changed accordingly. This is not possible in the original implementation of the simulation, as the rod element length is fixed to the distance between two spheres. To allow for variable rod element lengths, we implemented a new method to calculate the rod element length according to the number of rod elements n. The rod element length l is here calculated as

$$l = \frac{L}{n}, \tag{3.1}$$

where L is the total length of the guidewire. Note that the number of spheres is then simply $n + 1$. This way, the number of rod elements can be changed without changing the total length of the guidewire.

To additionally sustain the total weight of the guidewire M, the inverse masses of the spheres have to be adjusted accordingly. The mass of the i-th sphere is calculated as

$$m_i = \frac{M}{n+1}. \tag{3.2}$$

3.1.2 Collisions with Cylinders

The collision with cylinders is introduced in the framework by splitting the collision of the rod element onto the connected spheres. Instead of implementing a seperate collision constraint solver for the cylinders, we use the solver for the spheres. Upon a collision of the cylinder with the vessel wall, each of the both spheres attached to the rod element gets a collision registered. The normal of the collision, however, gets weighted by the distance of the contact point to the center of the rod element (projected onto the rod element orientation axis, i.e., only one-dimensional distances are measured. This means the normal vector is not normalized anymore.

Let $\mathbf{x}_i$ be the position of the i-th sphere, $\mathbf{c}$ the contact point of the collision of the i-th cylinder, and $\mathbf{n}$ the normal vector of the collision. First we define the distance vector $\mathbf{d}$ and the vector from the contact point to the center of the rod element $\tilde{\mathbf{c}}$ as

$$\mathbf{d} = \mathbf{x}_{i+1} - \mathbf{x}_i \tag{3.3}$$

and

$$\tilde{\mathbf{c}} = \mathbf{c} - \mathbf{x}_i \tag{3.4}$$

respectively. We then project the contact point onto the center line of the rod element, i.e., the connecting line between sphere i and $i+1$. The weighting w is hence simply given by

$$w = \frac{\tilde{\mathbf{c}} \cdot \mathbf{d}}{\|\mathbf{d}\|} = \cos\vartheta. \tag{3.5}$$

The collision with the cylinder is finally registered as usual sphere collisions, but the respective normal vector is weighted by the factor w or $1 - w$, depending on the sign of $\tilde{w} = w - 0.5$. This way, the collision constraint can be solved as

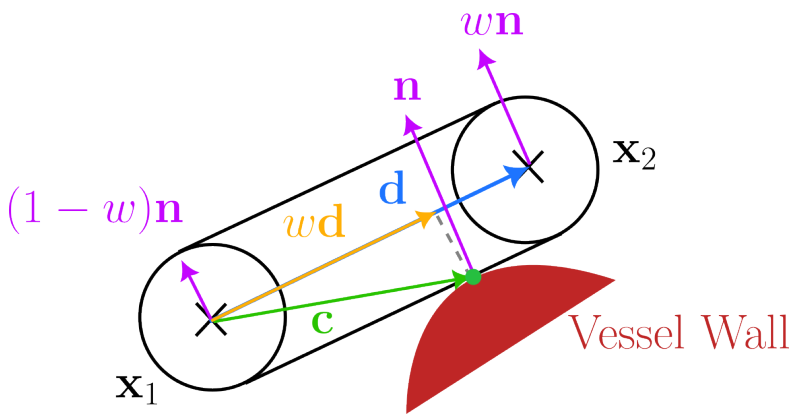

Figure 3.1: Illustration of the collision of a rod element with a cylinder. The collision is split onto the connected spheres, and the normal vector is weighted by the distance of the contact point to the center of the rod element.

sphere collisions, but still providing the angular momentum of the rod element. An illustration of the collision is shown in Figure 3.1.

3.1.3 Time Integration with Second-Order Verlet Scheme

The original implementation of the simulation used the explicit Euler method (See Section 2.2.1) in the prediction step. The Euler method is simple to implement and computationally cheap, but it is not very accurate. To improve the accuracy of the simulation, we implemented the second-order Verlet scheme to integrate the equations of motion for the new position prediction. The Verlet scheme is a symplectic integrator, which means that it conserves energy and momentum (See Section 2.2.2), i.e., the stability of the simulation is increased without increasing the computation time significantly. The algorithm scheme is implemented as follows: In the `predictionStep` class, the method `predictSpherePositions` calculates the new predictions for the sphere positions based the Verlet scheme Equation 2.7. However, this requires that the positions of the spheres at the previous time step are stored. This is done outside of the simulation loop. The Verlet scheme is not self-starting, so the initial positions of the spheres have to be calculated using the Euler method.

3.1.4 Steady State Simulation

The simulation of guidewires in the Unity game engine is a dynamic simulation, i.e., the guidewire is moving through the vessel. However, to judge the physical accuracy of the simulation, it is necessary to simulate the guidewire until it reaches a steady state. To achieve this with the Euler scheme, the velocity of the first sphere is simply not updated in the `UpdateSphereVelocities` method. The simulation is run for a few time steps until the guidewire reaches a new equilibrium state, i.e., a state where guidewire is not moving anymore, and the prediction error is minimal. The time the system needs to reach the steady state provides insights into the physical accuracy and convergence speed of the simulation.

However, the Verlet scheme does not require any velocities, so simply not updating the velocities is not possible. Instead, to achieve a steady-state simulation in case of the Verlet integration, the prediction step is skipped for the first sphere in the `predictSpherePositions` method. This way, the first sphere is not updated, and the simulation reaches a steady state.

3.1.5 Control of Guidewire

Instead of using external forces to push the guidewire through a vessel, in PBD it is also possible to simply displace the position of the spheres externally. Because the guidewire tries to minimize the constraints, it will start moving in the attempt to find a new stable state. In contrast to Viellieber [Vie23], who used external forces, Kreibich [Kre23] uses a constant displacement of the first sphere of the guidewire in one single, coordinate axis aligned, direction.

However, while working in simple cases, this method can sometimes 'pull' the guidewire through the vessel, which leads to undesired and unrealistic behavior. This is because the first sphere does not take part in the collision detection step. Instead, we propose to use a more sophisticated method to control the guidewire. In each time step, the position of the first sphere is displaced

by a small amount in the direction of the vessel center line. The displacement direction is calculated as the vector from the first sphere to the last sphere, i.e.,

$$\mathbf{d} = \frac{\mathbf{x}_2 - \mathbf{x}_1}{\|\mathbf{x}_2 - \mathbf{x}_1\|}. \tag{3.6}$$

Due to the stiffness of the guidewire, calculating the direction of the displacement in each time step mimics the behavior of a guidewire that is pushed through the vessel, leading to much more realistic and stable simulations.

3.2 Software Quality

To improve the software quality of the simulation, we implemented several software engineering practices. This was done to ensure the maintainability, extensibility, and reusability of the code. It enables other researchers to build upon the simulation and to easily understand the code.

3.2.1 JSON as Parameter Files

The simulation of guidewires in Unity requires a large number of parameters to be set. These parameters include the physical properties of the guidewire, such as the mass, the bending stiffness, and the damping, as well as the parameters of the simulation, such as the time step and the number of rod elements. To make it easier to set these parameters, we implemented a parameter handler class that reads the parameters from a JSON file. The JSON file contains all the parameters of the simulation, and the parameter handler class reads the parameters from the file and sets them in the simulation. This way, the parameters of the simulation can be easily changed by editing the JSON file, without having to change the code.

3.2.2 Parameter Handler Class

To make it easier to set the parameters of the simulation, we implemented a parameter handler class. The parameter handler class reads the parameters from

a JSON file and sets them in the simulation. The parameter handler class also contains methods to get and set the parameters of the simulation. This way, the parameters of the simulation can be easily accessed and changed by other classes.

3.2.3 Data Logger Class

In order to analyze the simulation results, it is necessary to log the data of the simulation. The data logger class writes the data of the simulation to a JSON file. The data logger class contains methods to log the positions and velocities of the spheres, elapsed wall time in milliseconds, as well as the time step and the number of iterations of the simulation. This way, the data of the simulation can be easily analyzed and visualized.

3.2.4 Command Line Handler

As the test environment is scripted in Python, the simulation is typically started from the command line. To allow easy access to variables from the command line, we implemented a command line handler class. The command line handler class reads the parameters of the simulation from the command line and sets them in the simulation. This way, the simulation can be easily run from the command line, without having to change the code.

3.2.5 Get Guidewire from Screen

Since the guidewire can be set with arbitrary numbers of rod elements, using a predefined guidewire, i.e., fixed length and number of spheres/rod elements, is not feasible anymore. Instead, the guidewire is defined by external parameters in the parameter handler class and constructed at the start of the simulation.

3.2.6 Update instead of FixedUpdate

Unity offers two different standard methods to update a simulation: `Update` and `FixedUpdate`. While `Update` is called once per frame, `FixedUpdate` is called at

a fixed time interval. `FixedUpdate` is used for physics calculations in games, as it is called at a fixed time interval, so it might be called once, twice, or not at all per frame. In the original implementation, the simulation was updated in `FixedUpdate`. However, as the simulation is not a game, but a numerical simulation, it is more appropriate to update the simulation in `Update`. The method `Update` is called once per frame, so the simulation is updated as fast as possible, the time step is always constant (which was not the case before) and the simulation speed is increased.

4 Results

The following chapter presents the results of this work. To measure the improvements of the guidewire simulation, discussed in out methods Chapter 3, we conducted different experiments. Similar to Kreibich [Kre23], we analyze the convergence of the system until it reaches a steady state, i.e., until the guidewire is not moving anymore after the initial displacement. The first sphere is fixed in position and velocity over the whole simulation, so its state is not changing. We start by analyzing the convergence of the guidewire in the longitudinal direction in free space, i.e., without any collisions, in the following Section 4.1. We then investigate the convergence of the guidewire in the transversal direction in free space in Section 4.2. Finally, we analyze the collision behavior of the guidewire with the blood vessel in Section 4.3.

4.1 Convergence to Steady State

As a main objective of the project was to improve the simulation speed while keeping the accuracy of the computations in physical meaningful bounds, we start by evaluating the convergence behavior of the guidewire to reach a steady state for different parameters after initial perturbation in the longitudinal direction.

4.1.1 Experimental Setup

The guidewire is initially placed in a straight line along the z-axis. The first sphere (with index $i = 0$) is displaced by a small amount in the z-direction, which we call the displacement Δz_0. The simulation is then run for a few seconds to observe the convergence of the guidewire. As the guidewire is a damped oscillator in the z-direction, it will oscillate around the initial position until it reaches a new equilibrium. We measure the elapsed time until the guidewire reaches the steady state, the in-game time needed to perform a sin-

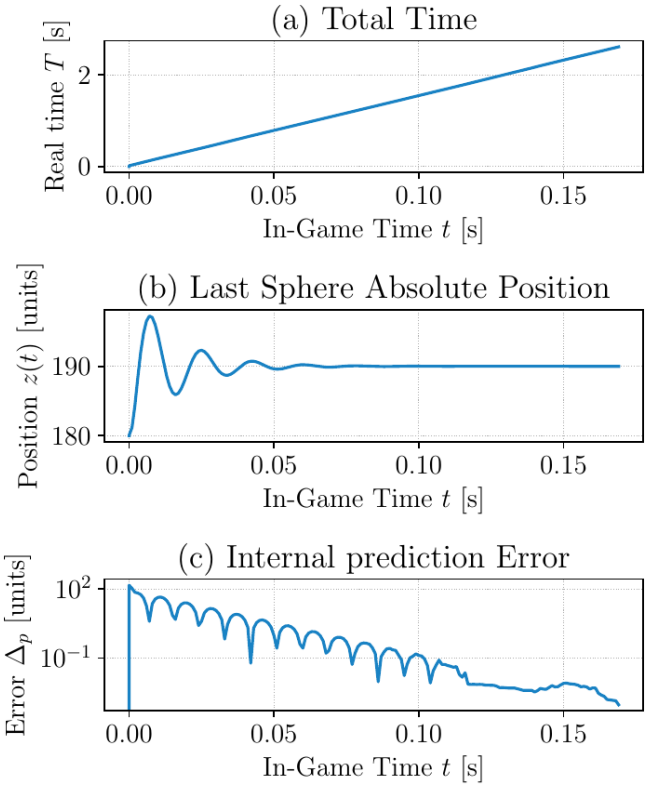

Figure 4.1: Time evolution of the last sphere in the guidewire simulation. The guidewire is pushed in the z-direction and the last sphere is observed. The simulation reaches a steady state after a few seconds.

gle time step, the position and velocity of the last sphere in the guidewire and the internal prediction error of the constraint solver. An example run of the guidewire simulation in free space showing convergence behavior can be seen in Figure 4.1.

In the first plot (a), we can see the relation between the in-game time and the real time. The *in-game time* is the time acquired by stepping in time discretely with time step size Δt. The *real time* is the physical time needed to perform the whole simulation and gives an indication of the computational time needed to evolve the system. Note, that this relationship is not necessarily linear and depends, e.g., on the scheduling of the CPU. The second plot (b) shows the absolute position of the last sphere in the guidewire over time in z-direction. The last plot (c) shows the internal prediction error Δ_p of the last sphere in the guidewire

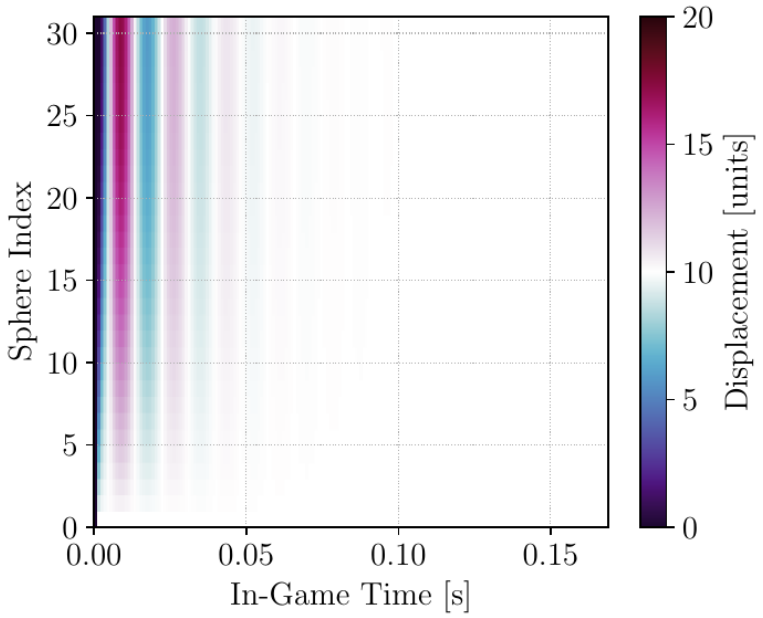

Figure 4.2: Temporal evolution of the last sphere in the guidewire simulation. The guidewire is pushed in the z-direction and the last sphere is observed. The simulation reaches a steady state after a few seconds.

over time. The prediction error Δ_p is the difference between the predicted position and the actual position of the last sphere, and can therefore be used as a measure of the accuracy of the constraint solver. The smaller the prediction error, the better the constraints are satisfied. Together with the relaxation time of the damped oscillator, the prediction error Δ_p gives an indication of the convergence of the simulation. Only if Δ_p drops below a certain threshold $\epsilon \ll 1$ units, the simulation has reached its steady state after the initial push. In the following, we always use $\epsilon = 0.001$ units.

As we can see in Figure 4.1, the guidewire reaches a steady state after a few seconds. The oscillations of the guidewire are damped and the prediction error is small, indicating that the constraints are satisfied at the end of the simulation. Furthermore, the time-evolution of the displacement of all spheres in the same experiment is illustrated in Figure 4.2, where we can see the collective oscillatory behavior of the guidewire until it reaches the steady state, i.e., the

Fixed Parameter	**Value**
Radius of spheres r	5 units
Diameter of sphere d	$2r = 10$ units
Total length L	300 units
Rod element length l	$L/n = 10$ units
Stretch stiffness parameter k_A	0.1
Bend stiffness parameter k_B	0.1
Collision stiffness parameter k_C	10^{-4}
Variable Parameter	**Value**
Constraint solver steps m	500
Number of rod elements n	30
Number of spheres	$n + 1$
Displacement Δz_0	10 units
Time step size Δt	0.01 s
Total mass M	0.01 units

Table 4.1: Default parameters used in the guidewire simulation.

oscillations are not independent of each other. This will be different when we consider the transversal perturbation of the guidewire in the Section 4.2.

The experiments are repeated for different parameters, such as the amount of constraint solver steps m, the number of rod elements n, the displacement of the first sphere Δz_0, the time step size Δt. The default parameters are set according to Table 4.1, which are corresponding to the result shown in Figure 4.1 and Figure 4.2. We used $n = 30$ as the default number of rod elements, because in this case $n \times d = L$, i.e., the spheres are one next to the other, but not overlapped, which is the minimum. Kreibich [Kre23] worked only with $n > 30$ at total length of the guidewire of $L = 300$, where spheres are overlapping. Achieving numbers of rod elements below $n = 30$ is amongst the main contributions of this work. In the following experiments, we only change the selected variable parameters, keeping the other parameters by default.

4.1.2 Convergence Analysis

In order to give an external estimation about the convergence of the simulation, we calculate the distance of the expected position $\tilde{\mathbf{x}}_i$ and the actual positions

$\mathbf{x}_i(t)$ of the spheres in the guidewire. In case of the longitudinal perturbation in free space, the expected position can be calculated analytically as

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i(t=0) + (0, 0, \Delta z_0)^\top, \tag{4.1}$$

where $\mathbf{x}_i(t=0)$ is the initial position of the i-th sphere and Δz_0 is the displacement of the first sphere in z-direction. The displacement error $\Delta_{\mathbf{x},i}$ of the i-th sphere is then the distance between the expected theoretical position after displacement and the actual position of the spheres, i.e.,

$$\Delta_{\mathbf{x},i}(t) = \|\mathbf{x}_i(t) - \tilde{\mathbf{x}}_i\|_2, \tag{4.2}$$

where $i \in \{0, ..., n\}$ is the index of the i-th sphere and n the total number of rod elements. The number of spheres is then simply $n+1$.

The time to reach convergence varies with different parameters, such as number of constraint solving steps or time step size. Using the displacement error $\Delta_{\mathbf{x},n+1}(t)$ of last sphere over time, we can estimate the relaxation time of the damped oscillator as the relaxation time τ of an exponential decay. As the push of the guidewire in this experiment is only along the z-direction, the system can be described by a one-dimensional function. The local maxima $\xi(t)$ of the displacement error of the last sphere $\Delta_{\mathbf{x},n+1}(t)$ can therefore be approximated as

$$\xi(t) = \xi_0 \exp(-t/\tau), \tag{4.3}$$

where ξ_0 corresponds to the amplitude of the oscillation envelope, and τ is the desired relaxation time of the damped oscillator.

We find each peak of the oscillation and fit Equation 4.3 to the peaks. The relaxation time τ can then be plotted as a function of other parameters. An example of such a fit can be seen in Figure 4.3.

Additionally, we measure the total error of the guidewire at the end of the simulation, i.e., the difference between the expected theoretical positions after displacement $\tilde{\mathbf{x}}_i = \mathbf{x}_i(t=0)$ and the actual position of the spheres $\mathbf{x}_i(t=t_{\text{end}})$

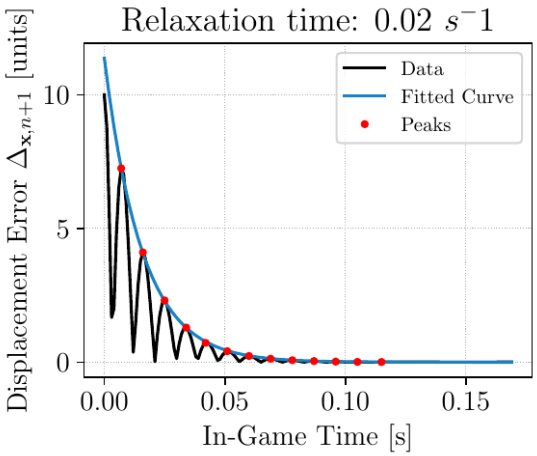

Figure 4.3: Example of a fit of the relaxation time τ to the peaks of the last sphere in the guidewire simulation. The relaxation time is the decay time of the exponential decay.

at the end of the simulation. Using Equation 4.2, the total error can be expressed as

$$\Delta_{\mathbf{x}} = \sum_{i=0}^{n} \Delta_{\mathbf{x},i}(t = t_{\text{end}}). \tag{4.4}$$

The total error $\Delta_{\mathbf{x}}$ can also be plotted as a function of the different parameters. An example of the displacement error at the end of an experiment as a function of the sphere index i can be seen in Figure 4.4. Here we can see that the displacement error is increasing with the sphere index, with the first sphere having no error at all. This is expected, as the first sphere is fixed in position and velocity over the whole simulation and the error sums up over the rod elements. However, the error is very small compared to the initial displacement of the first sphere ($0.0049/10 \ll 1$), indicating that the constraints are satisfied at the end of the simulation and we can neglect the error of the constraints.

4.1.3 Constraint Solver Steps

The amount of constraint solver steps has one of the most severe influences on the convergence time and physical correctness of the guidewire simulation. The more steps, the better the constraints are satisfied. However, better ap-

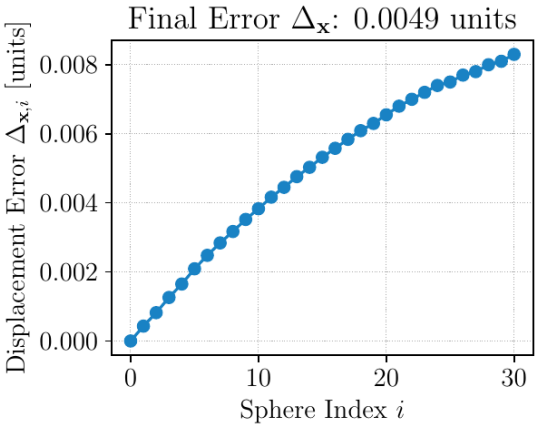

Figure 4.4: Displacement error of the spheres at the end of the guidewire simulation. The error is the difference between the expected theoretical positions after displacement and the actual position of the spheres.

proximations require also more computation time. To analyze this behavior, we conducted an experiment where we varied the amount of constraint solver steps from 100 to 1000. The results of this experiement can be seen in Figure 4.2. As expected, the relaxation time τ (Figure 4.5 (a)) decreases with the amount of constraint solver steps. The better approximation of the constraints leads to an increase of the stiffness parameter k_A, k_B, and k_C, which also decreases the time needed to reach the steady state. The average step time (Figure 4.5 (b)) needed to reach the steady state, on the other hand, increases with the amount of constraint solver steps. This increase is approximately linear, as the computation time is proportional to the amount of constraint solver steps. Contrary to our expectation, the In-Game time (Figure 4.5 (c)) of the spheres decreases. This can be explained by the fact, that the relaxation time decreases faster than the time step size. In combination, the computation time (Figure 4.5 (d)) stays relatively constant with increasing constraint solver steps. This indicates, that the constraints are not the limiting factor in the simulation.

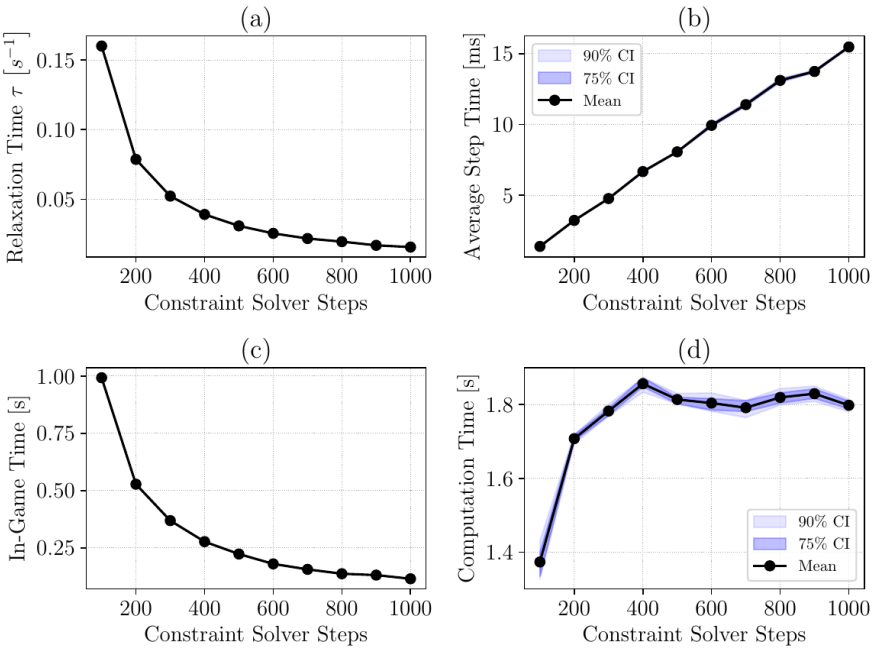

Figure 4.5: Convergence analysis of the guidewire simulation as a function of the amount of constraint solver steps. (a) Relaxation time τ, (b) Average execution time of a single simulation step, (c) Total in-game time needed to reach the steady state, (d) Total computation time needed to reach the steady state.

4.1.4 Number of Rod Elements

The additional implementation of collisions with the cylinders makes it now possible to use rod element lengths l which are longer than two times the radius of the spheres, i.e., $l > 2r$. This is important for the real-time application of the simulation, because less spheres (and rods) need much less computation time. This is shown in Figure 4.6 (b). The more rod elements, the larger the average execution time for a single loop. However, as can be seen from Figure 4.6 (b), the relaxation time τ is only influenced by a small amount, but increasing with the number of rod elements. This is because the rod elements are only used to approximate the guidewire, and the constraints are only applied to the spheres. Since the relaxation time is only slightly influenced by the number of rod elements, the total in-game time needed to reach the steady state (Figure 4.6

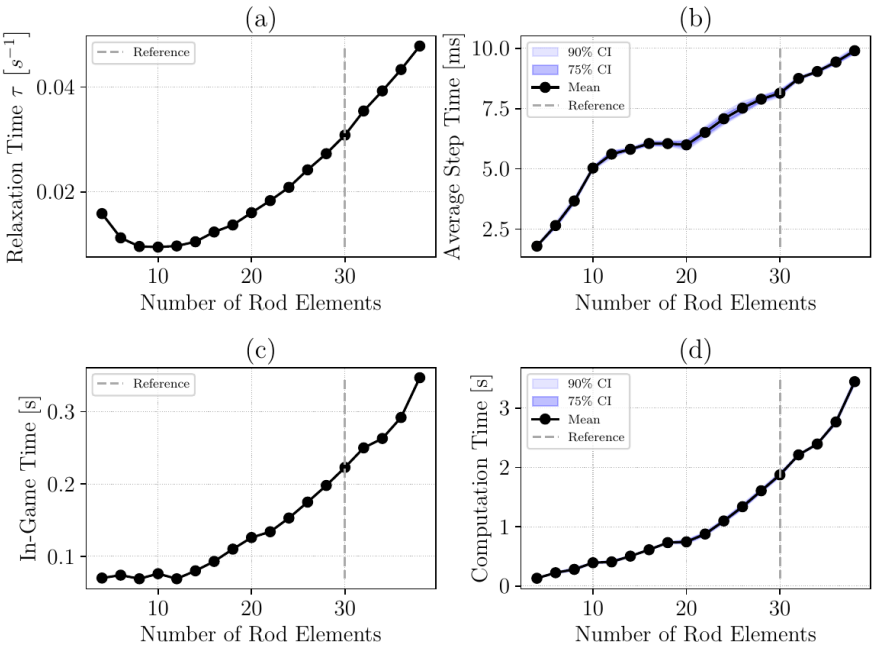

Figure 4.6: Convergence analysis of the guidewire simulation as a function of the number of rod elements. (a) Relaxation time τ, (b) Average execution time of a single simulation step, (c) Total in-game time needed to reach the steady state, (d) Total computation time needed to reach the steady state. The reference solution with $n = 30$ rod elements is shown as a dashed line.

(c)) and the total computation time needed to reach the steady state (Figure 4.6 (d)) are increasing with the number of rod elements.

4.1.5 Displacement

The amount of displacement applied to the first sphere has barely any influence on the total time needed to reach the steady state, nor does it substantially change the relaxation time (Figure 4.7 (a)-(d)). The reason for this can be reduced to the efficient constraint solver, which is able to solve the constraints in a few steps. Important to note is, that the simulation is still very stable, even with large displacements of the first sphere, which makes it attractive for real-time applications.

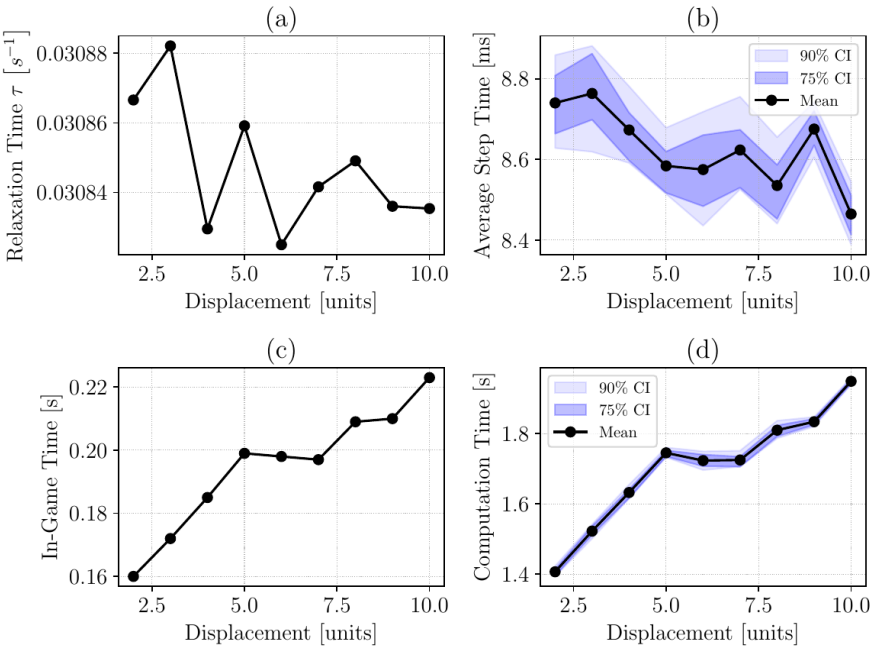

Figure 4.7: Convergence analysis of the guidewire simulation as a function of the displacement of the first sphere. (a) Relaxation time τ, (b) Average execution time of a single simulation step, (c) Total in-game time needed to reach the steady state, (d) Total computation time needed to reach the steady state.

4.1.6 Time Step Size

The most important parameter in a simulation of a physical system is the time step size Δt. This influences how accurate the predictions of the derivatives in the Euler-method or the Verlet scheme can be approximated. On the other hand, if Δt is very small, the computation time needed to evolve the system gets very large.

This behavior could be confirmed by measuring the average time it takes to perform a single time step. The results of this experiment can be seen in Figure 4.8.

As expected, the relaxation time τ increases with the time step size Δt perfectly linear. This is because per time interval, the system uses much more constraint solver steps. This leads to a faster convergence of the system with small time steps. This way, the average step time and the total computation time stay

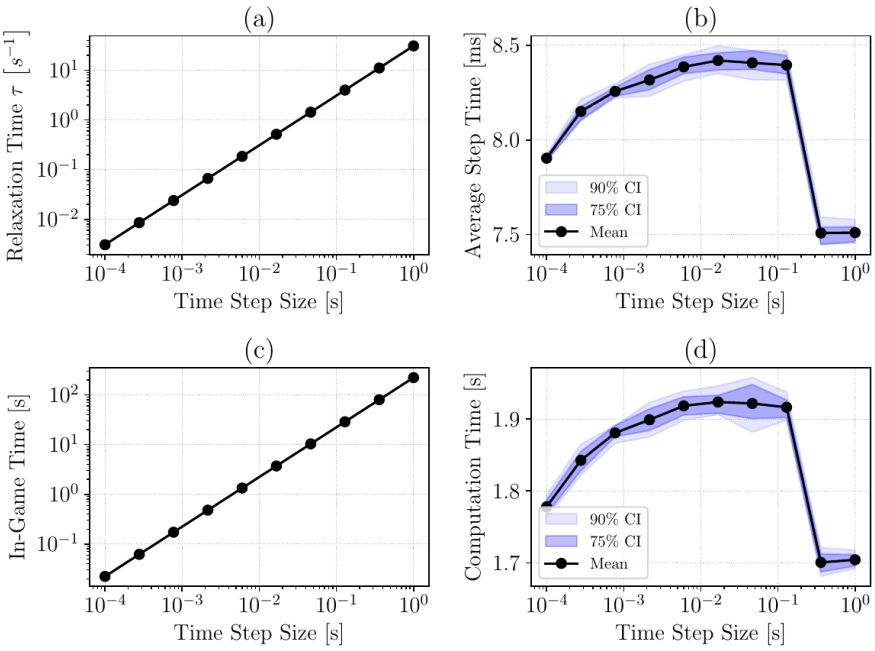

Figure 4.8: Convergence analysis of the guidewire simulation as a function of the time step size. (a) Relaxation time τ, (b) Average execution time of a single simulation step, (c) Total in-game time needed to reach the steady state, (d) Total computation time needed to reach the steady state.

relatively constant, while the total in-game time needed to reach the steady state increases with the time step size. This can be seen in Figure 4.8 (a)-(d).

4.1.7 Total Mass

The inertia of the guidewire is determined by the total mass M of the spheres. The more mass, the more inertia the guidewire has, and the longer it takes to reach the steady state. This can be seen in Figure 4.9 (a). However, contrary to the prior cases, here it has a significant influence on the total computation time needed to reach the steady state (Figure 4.9 (d)). This is because the constraint solver has to solve the constraints for a longer time, as the system is oscillating slower. The total in-game time needed to reach the steady state (Figure 4.9 (c)) is also increasing with the total mass, but not as much as the total computation time. The average step time (Figure 4.9 (b)), on the other hand, is not changing

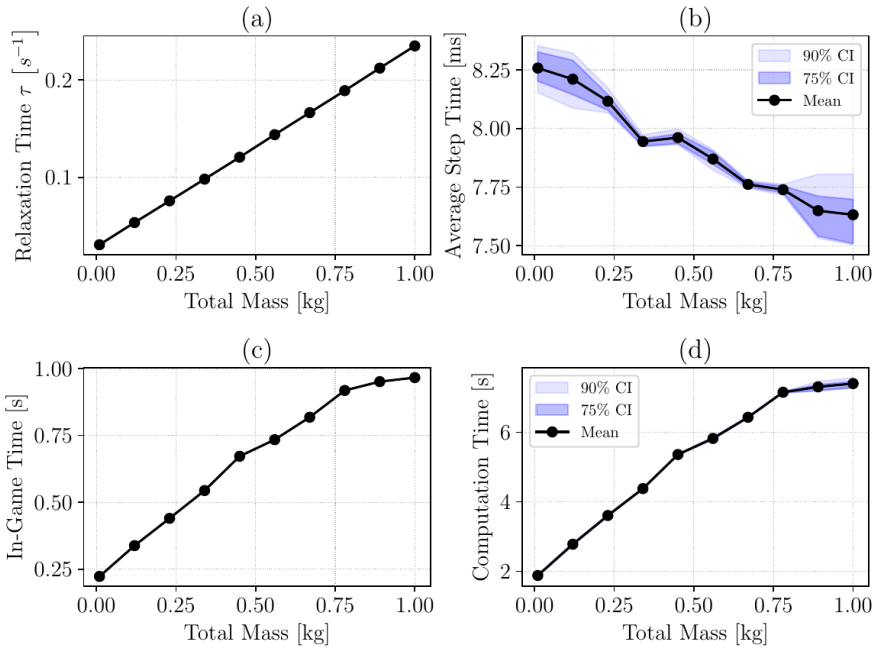

Figure 4.9: Convergence analysis of the guidewire simulation as a function of the total mass of the guidewire. (a) Relaxation time τ, (b) Average execution time of a single simulation step, (c) Total in-game time needed to reach the steady state, (d) Total computation time needed to reach the steady state.

significantly with the total mass. This is because the step time is proportional to the amount of constraint solver steps, and not to the total mass of the guidewire.

4.2 Transversal Perturbation

The first experiment focused only on the longitudinal waves of the guidewire, which are realized by the stretch constraint of the cosserat rod model. In the following, we consider the transversal waves along the guidewire, achieved by the bend-twist constraint. This is achieved by a small displacement Δy_0 of the first sphere along the y-direction, i.e, similar to before the displacement of the first sphere is $\Delta \mathbf{x}_0 = (0, \Delta y_0, 0)^\top$.

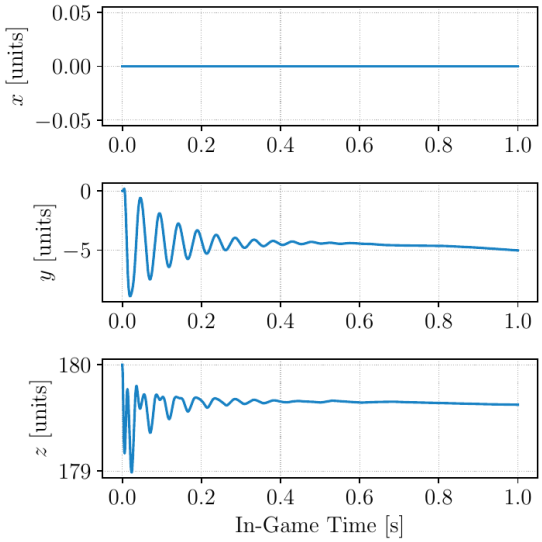

Figure 4.10: Reference simulation for the convergence and error analysis of perturbations of the guidewire in the transversal direction. Shown are the time evolution of the last sphere in the x, y and z-direction.

As the movement in the transversal direction is much harder to predict, we do not measure the relaxation time. The expected positions $\tilde{\mathbf{x}}_i$ can not be calculated analytically, so we define the expected position by a reference simulation with a large amount of constraint solver steps, many rod elements and a very small time step size. We then measure the evolution of the guidewire with different parameters and compare it to the reference solution. The time-evolution of the last sphere of the reference solution can be seen in Figure 4.10. As it is apparent from the position data of the last sphere, the movement is a transversal wave in the y-z-plane, i.e, there is no movement of the guidewire in the x-direction. As the medium of the guidewire is oscillating, it is resonant to certain frequencies. This leads to the development of a standing wave with one open end (the last sphere) and a closed end (the first sphere). The standing wave is a superposition of two waves traveling in opposite directions.

The experiments are repeated for different parameters, such as the amount of constraint solver steps, the rod element length, the displacement of the first sphere, the time step size. We use the same default parameters as in the lon-

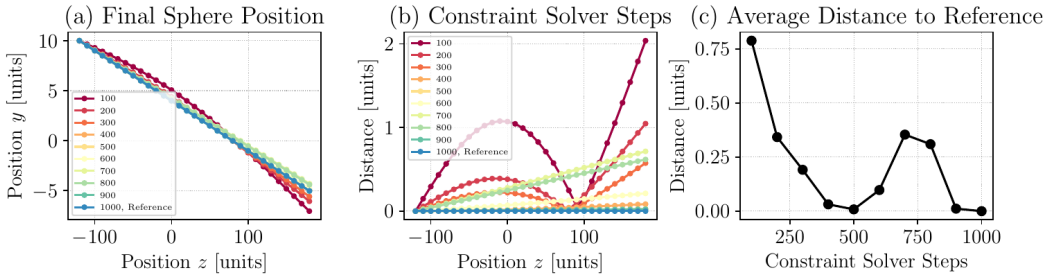

Figure 4.11: Transversal perturbation accuracy analysis of the guidewire simulation as a function of the amount of constraint solver steps. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

gitudinal perturbation (Table 4.1). Similar to before, we measure the distance of the expected position $\tilde{\mathbf{x}}_i$ and the actual positions $\mathbf{x}_i(t)$ of the spheres in the guidewire. The displacement error $\Delta_{\mathbf{x},i}$ of the i-th sphere and the total error $\Delta_{\mathbf{x}}$ is then calculated as in Equation 4.2 and Equation 4.2, respectively.

4.2.1 Constraint Solver Steps

Also in the transversal case, the constraint solver steps have a huge influence on the accuracy of the simulation. The more steps, the better the constraints are satisfied. The absolute position of the guidewire spheres at the end of the simulation for different constraint solver steps can be seen in Figure 4.11 (a) The final distance of the guidewire to the reference solution is displayed Figure 4.11 (b). Contrary to our expectation, the average distance to the reference solution (Figure 4.11 (c)) has a small increase in the range of 700 constraint solving steps. This could be due to resonances in the system, as the constraint solving steps have a direct influence on the stiffness parameters k_A, k_B and k_C.

4.2.2 Time Step Size

The time step size has less influence on the accuracy of the simulation in the transversal case. The distance of the last sphere to the reference solution is al-

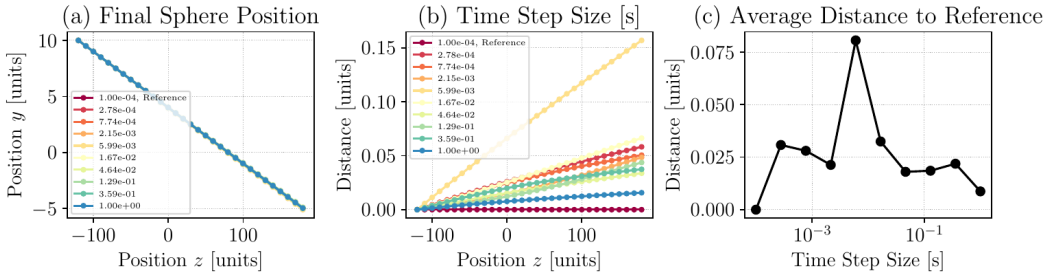

Figure 4.12: Transversal perturbation accuracy analysis of the guidewire simulation as a function of the time step size. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

most constant for different time step sizes, as can be seen in Figure 4.12 (c). There is only a small increase in the average distance to the reference solution for a time step size of 10^{-2} s. Similar to the constraint solving steps, this could be due to resonances.

4.2.3 Number of Rod Elements

The number of rod elements has a significant contribution to the accuracy of the simulation. The more rod elements, the better the approximation of the guidewire. The distance of the guidewire to the reference solution is almost constant for different rod element lengths, as can be seen in Figure 4.13 (b). Also, this time, some kind of resonance can be observed in the average distance in Figure 4.13 (c), but this time with more than one peak.

4.2.4 Total Mass

In our experiments, the total mass of the guidewire has one of the greatest influences on the accuracy of the simulation. After initial displacement, guidewires with different total masses show vastly different behavior. This can be seen in Figure 4.14 (a) and (b) as large oscillations of the guidewire. The average distance of the guidewire to the reference solution is oscillating with the total mass, with 0.01 being the reference solution. This can be seen in Figure 4.14 (c).

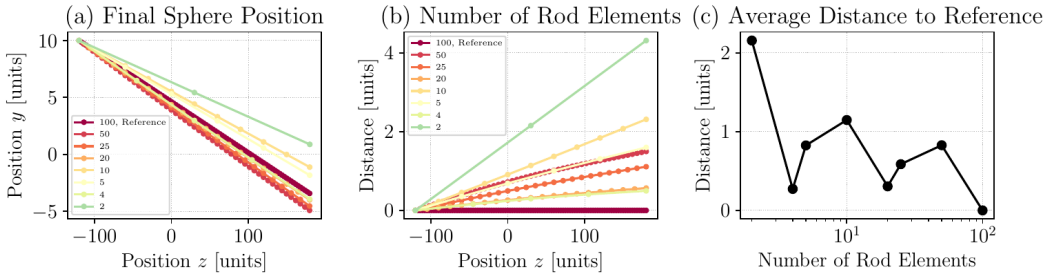

Figure 4.13: Transversal perturbation accuracy analysis of the guidewire simulation as a function of the number of rod elements. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

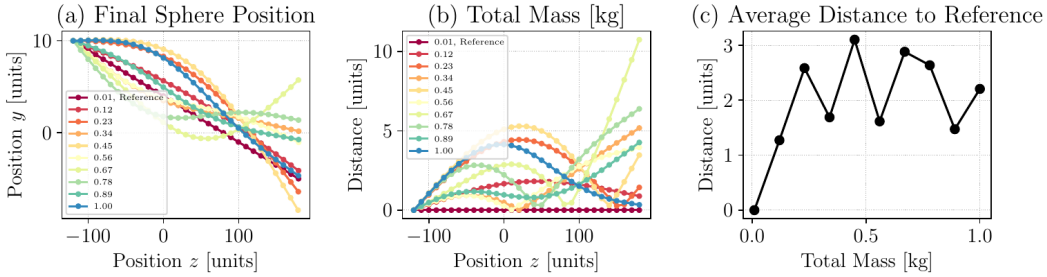

Figure 4.14: Transversal perturbation accuracy analysis of the guidewire simulation as a function of the total mass of the guidewire. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

4.3 Collisions

To finally investigate the collision behavior of the guidewire with the blood vessel, in the following experiments we placed the guidewire right before the vessel wall. We displace the first sphere in the z-direction, such that the guidewire collides with the wall. We again analyze the convergence behavior of the last sphere, however, the movement is now much harder to predict. In this scenario, it makes no sense to measure the relaxation time, as this is vastly influenced by the geometry of the blood vessel. Instead, we measure the evolution of a guidewire with a large amount of constraint solver steps and a very small time

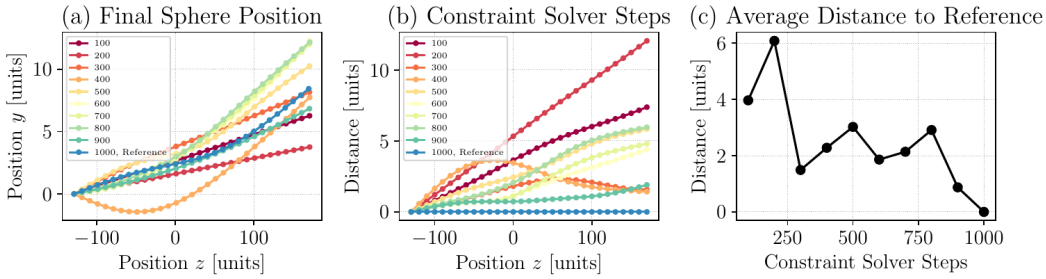

Figure 4.15: Collision accuracy analysis of the guidewire simulation as a function of the amount of constraint solver steps. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

step size and use this measurement as a reference, similar to the transversal case from before. We want to analyze at which parameters the simulation diverges from the reference solution.

4.3.1 Constraint Solver Steps

In the collision experiment, the constraint solver steps shows similar behavior as in the longitudinal and transversal perturbation. The more steps, the better the constraints are satisfied. This can be seen in Figure 4.15 (a) and (b). The average distance of the guidewire to the reference solution decreases for increasing constraint solver steps, as can be seen in Figure 4.15 (c).

4.3.2 Time Step Size

The time step size shows a much clearer picture for the collision experiment. The distance of the guidewire to the reference solution is increasing with the time step size, but is nearly zero for time steps from 10^{-4} to 10^{-3} s (see Figure 4.16). This means that the simulation is very sensitive to the time step size, and the time step size has to be chosen very small to achieve a good approximation of the guidewire.

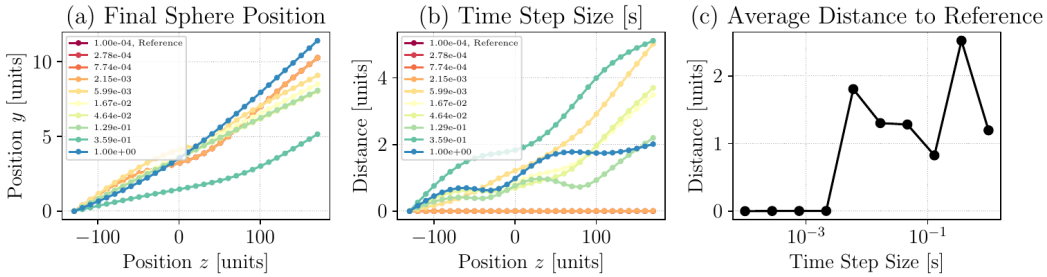

Figure 4.16: Collision accuracy analysis of the guidewire simulation as a function of the time step size. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

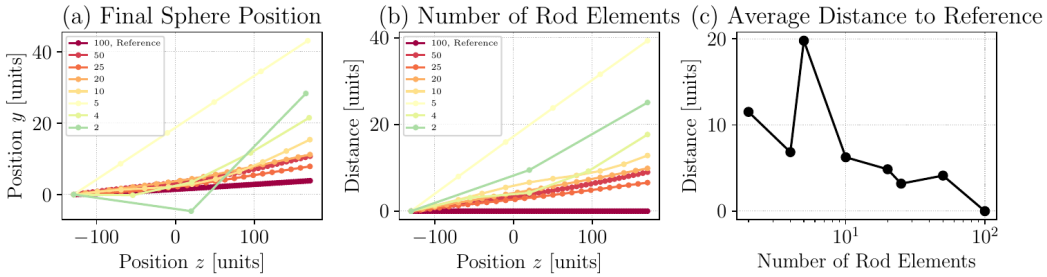

Figure 4.17: Collision accuracy analysis of the guidewire simulation as a function of the number of rod elements. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

4.3.3 Number of Rod Elements

Colliding with the blood vessel, the number of rod elements has a huge influence on the accuracy of the simulation. The more rod elements, the better the approximation of the guidewire. The distance of the guidewire to the reference solution is almost linear for different rod element lengths, as can be seen in Figure 4.17 (a) and (b). For the collision case, some kind of resonance can be observed in the average distance in Figure 4.17 (c), but generally a decreasing trend with increasing number of rod elements.

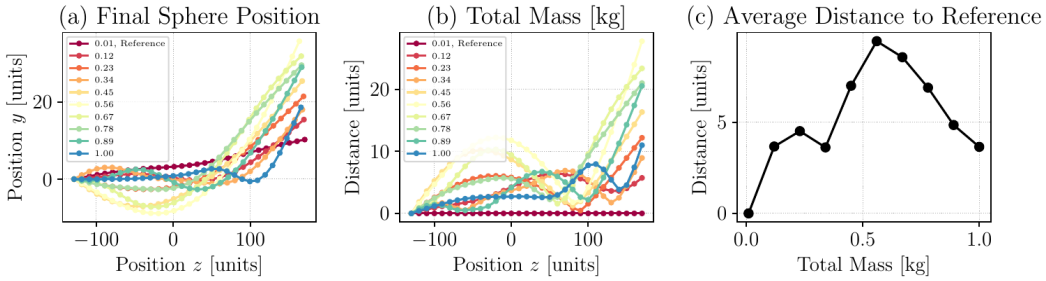

Figure 4.18: Collision accuracy analysis of the guidewire simulation as a function of the total mass of the guidewire. (a) Position of the guidewire spheres in the yz-plane, (b) Distance of the guidewire the reference solution at the last time step, (c) Average distance of the guidewire to the reference solution over the whole time.

4.3.4 Total Mass

Finally, the mass influences the behavior of the guidewire in collision experiment significantly. Here, the distance seems to be oscillating more with increasing mass, leading to a higher average distance to the reference solution. This can be seen in Figure 4.18 (a) and (b). Contrary to the other experiments, very high masses, i.e., > 0.5 kg, show a decreasing trend in the average distance to the reference solution (see Figure 4.18 (c)). This could be also explained as a resonance in the system, as the guidewire is oscillating in the blood vessel.

5 Discussion

In this work, we have successfully improved the simulation of guidewires in Unity employing the Cosserat rod model. We explored the foundational aspects of position and orientation-based dynamics, which serve as the core of our simulation framework. Additionally, we delved into the theory of time integration in physical simulations, examining both the Euler and Verlet integration methods. In this chapter, we will review the outcomes of our practical implementation and discuss potential avenues for further improving the simulation.

Position-based dynamics (PBD) has proven to be an efficient approach for simulating soft bodies, offering both stability and speed. Our guidewire simulation relies on the PBD Cosserat rod model, which, despite its effectiveness, has certain limitations. One significant drawback is that the simulation's physical accuracy is compromised due to the iterative nature of constraint solving. This iterative process can result in slow convergence, particularly in complex scenarios. Moreover, the original formulation using the Euler method introduces errors proportional to the time step Δt, and it fails to conserve the system's energy, potentially leading to numerical instabilities. These instabilities may manifest as high-frequency oscillations, which can destabilize the simulation.

The introduction of the Verlet integration scheme has demonstrated notable improvements in stability and accuracy compared to the Euler method. Verlet integration is a symplectic integrator, meaning it conserves the system's energy, with errors proportional to the square of the time step Δt^2, classifying it as a second-order method. Our implementation of the Verlet integration scheme has also further enhanced stability, with the by effectively suppressing high-frequency oscillations and accelerating the simulation's convergence.

In our current methods, we employed the Verlet integration scheme solely for predicting the positions of the particles. However, applying Verlet integration to both the positions and orientations of the rod elements in POBD could yield even greater accuracy. This is an avenue worth exploring in future work.

Looking forward, one promising direction for improving the simulation is the implementation of Extended Position-Based Dynamics (XPBD), as proposed by Macklin et al. [Mac+16]. XPBD enhances the accuracy of simulations by incorporating a more sophisticated constraint solver. It addresses the limitations of PBD by eliminating the dependence of relaxation time on the time step size and the number of constraint solver iterations through the introduction of compliance α, which represents the inverse stiffness of the constraints. Additionally, XPBD suggests that using sub-steps with smaller time increments, rather than larger steps with numerous constraint-solving iterations, can lead to even faster convergence.

To fully realize the potential of the Cosserat rod model in guidewire simulations, future research should also consider the twisting behavior of the guidewire, a common technique utilized by physicians during actual procedures. Incorporating this aspect could provide a more comprehensive and realistic simulation, further bridging the gap between virtual training and real-world medical practice.

6 Conclusion

In this work, we have enhanced the simulation of guidewires for medical interventions by incorporating the Position-Based Dynamics (PBD) framework and the Cosserat rod model, replacing the implicit Euler integration with the symplectic, second-order Verlet integration scheme. The simulation was implemented within the Unity game engine, where we conducted convergence and accuracy studies to assess its physical usability and stability. Additionally, we explored the theoretical underpinnings of time integration in physical simulations and compared the performance of the Euler and Verlet integration methods.

Our results indicate that the Verlet integration scheme offers superior stability and accuracy compared to the Euler method. However, despite this improvement in the integration technique, the overall convergence of the simulation has not significantly improved. The primary limitation remains the iterative nature of constraint solving, which can slow down the simulation's convergence and reduce its physical accuracy.

Two critical parameters in the simulation are the time step size Δt and the number of constraint solver iterations. Selecting an appropriate time step size is crucial; a step size that is too small may cause slow convergence, while one that is too large can introduce numerical instabilities. Similarly, the number of constraint solver iterations must strike a balance between accuracy and performance. Too few iterations can hinder convergence, while too many can degrade the simulation's performance.

The second most significant improvement in the simulation was the introduction of collision handling with cylindrical objects. This enhancement has optimized the simulation by reducing the number of spheres and rod elements required to represent the guidewire, thereby minimizing computational overhead.

Apart from the simulation improvements, we implemented software engineering best practices enhancing the codebase's maintainability and extensibil-

ity. By adhering to object-oriented design principles and employing a modular architecture, we have ensured that the simulation code is easy to understand, modify, and extend.

In summary, our work has contributed to the ongoing development of guidewire simulations for medical interventions by enhancing the simulation's stability and accuracy. By continuing to refine and optimize the simulation, we aim to create a valuable tool for training physicians in complex medical procedures, ultimately improving patient outcomes and advancing medical education.

Bibliography

[Ken14] B. Kenwright. "Position-Based Dynamics (e.g., Verlet System)". *The Path To Working Smarter Not Harder*, 2014 (cit. on p. 7).

[Kre23] A. S. Kreibich. "Convergence Analysis of Guidewire Simulations Employing Real Patient Data". *Universität Heidelberg*, 2023 (cit. on pp. 5, 11, 15, 18).

[KS16] T. Kugelstadt and E. Schömer. *Position and Orientation Based Cosserat Rods*. Eurographics/ ACM SIGGRAPH Symposium on Computer Animation. 2016. URL: https://diglib.eg.org/handle/10.2312/sca20161234 (visited on 07/30/2024). Pre-published (cit. on pp. 1, 5).

[Mac+16] M. Macklin, M. Müller, and N. Chentanez. "XPBD: Position-Based Simulation of Compliant Constrained Dynamics". In: *Proceedings of the 9th International Conference on Motion in Games*. MiG '16: Motion In Games. ACM, Burlingame California, 10, 2016, pp. 49–54. ISBN: 978-1-4503-4592-7. URL: https://dl.acm.org/doi/10.1145/2994258.2994272 (visited on 07/30/2024) (cit. on p. 35).

[Mül+07] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. "Position Based Dynamics". *Journal of Visual Communication and Image Representation* 18:2, 2007, pp. 109–118. ISSN: 10473203. URL: https://linkinghub.elsevier.com/retrieve/pii/S1047320307000065 (visited on 07/30/2024) (cit. on pp. 1, 3).

[Uni23] Unity Technologies. *Unity*. Version 2023.2.3. Game development platform. 2023. URL: https://unity.com/ (cit. on p. 5).

[Vie23] R. Viellieber. "Simulating Guidewires in Blood Vessels Using Cosserat Rod Theory". *Universität Heidelberg*, 2023 (cit. on pp. 1, 5, 11).