DEVELOPMENT OF TECHNIQUES FOR MODELING THE STATIC BUCKLING OF EULER BEAM AND DYNAMIC RESPONSE OF KIRCHHOFF RODS: APPLICATION TO SURGICAL SIMULATION AND TRAINING

A Dissertation

by

ZHUJIANG WANG

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Arun Srinivasa
Committee Members,	Sevan Goenezen
	Krishna Narayanan
	Anastasia Muliana
Head of Department,	Andreas Polycarpou

May 2016

Major Subject: Mechanical Engineering

ABSTRACT

In this dissertation, we present novel schemes for a static simulation of a buckled Euler beam with curve channel constraints in two dimensional space and simulation of the dynamic response of a soft Kirchhoff rod in three dimension space at real time rate. The aim of this model is to provide a robust and fast means for simulating endoscopes and surgical threads for training and surgical simulation purposes. Finding a static configuration of a buckled cantilever elastic beam constrained in a curved solid channel subject to end forces is a simple model of endoscopy and it is posed as the minimization of an energy functional. We solve it by a novel technique, a variant of a dynamic programming approach called the Viterbi algorithm. The core idea of this approach is to discretize the variables describing the potential energy and to construct a set of admissible configurations of the beam. The Viterbi algorithm is then employed to search through the set of possible beam configurations and locate the one with the minimum potential energy in a very computationally efficient way. The new approach does not require any gradient computations and could be considered as a direct search method, and thus can be guaranteed to find the global minimum potential energy. Also the constraints can be automatically satisfied by constructing the proper set of all the possible configurations. The approach can also be used to find feasible starting configurations associated with conventional minimizing algorithms.

We also discuss a novel scheme based on discrete variational integrators to study the dynamics of an inextensible thin Kirchhoff rod which is a model for a surgical thread. The benefits of such approach are that it is a very efficient scheme that guarantees conservation of momentum and energy over very long times so that a real time simulator can be operated over long periods of time. In addition, we report on an innovative technique to capture the inextensibility as well as the internal dissipation of the rod efficiently. Finally, a new collision avoidance scheme based on a continuous penalty force is employed to simulate the interaction of the rod with the surrounding medium. The simulations performed capture the formation of plectoneme, i.e. a loop of helices twisted together. Lastly, the scheme is employed to simulate the tying of a square knot. This model can be used to simulate surgical threads at real time rate.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Arun Srinivasa. You are a tremendous mentor for me. You have always taught me how to think and do research in a philosophical way, which I could never have learned from any books or research papers. I have learned a lot from you and it will benefit my entire life.

Also, I really appreciate the assistance, the valuable discussion and the support of my research, from Dr. Annie Ruimi, Dr. Marco Fratarcangeli, and Dr. Krishna Narayanan. Additionally, I thank Dr. Alan Freed, Dr. Sevan Goenezen, and Dr. Anastasia Muliana for serving as my committee members and giving me suggestions to improve my research.

I would like to acknowledge the NPRP award from the Qatar National Research Fund for the financial support of my research.

Thank to all my friends for bringing me so much happiness. I have lived a happy life for the last four years in College Station. A special thanks to my family members for the support of my study in the US. Words cannot express my feelings, nor my sincere gratitude to my mom, father, and my grandpa. I especially thank my wife, Elwing Yang. With you in my life, the firefly in the dark is the sun in my heart.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Motivation

Among many surgical tasks, suturing and knotting are considered particularly challenging for medical school students. It is not atypical for a student to take up to one minute to tie a knot compared to the five or seven knots per minute an experienced surgeon will be able to achieve and at that rate, the four hundred knots needed for an organ surgery will require six hours of surgical time devoted to that. Achieving a high level of competency in suturing requires a significant amount of practice. However, there is a consensus in the medical community that the traditional surgical training methods that rely on practicing on animals or human corpse or on plastic and foam models have limitations and have become slightly obsolete in the light of the 21th century (see Figure 1.1). These traditional methods are known to have limitations because [1]: i) dead tissues do not exhibit the same response as healthy ones, ii) humans and animals have different physiologies, iii) plastic or foam models lack realism, iv) novice trainees have a higher risk of being exposed to contaminated blood while manipulating sharp instruments, v) experiments on animal raise ethical considerations and vi) students of Muslim or Jewish faith may have religious conflicts in practicing on porcine tissues. In addition, there has been raising pressure from animal right activists and religious groups having conflicts touching porcine tissues [1].

Moreover, advanced surgery techniques, such as robotic surgeries and minimally-invasive surgeries (MIS) as shown in Figure 1.2 and 1.3, have played a more and more important role in modern medicine. According to Dr. Fabrizio Michelassi, MD [2], Surgeon-in-Chief at NewYork-Presbyterian Hospital / Weill Cornell Medical Center, "Modern surgery is minimally invasive, and that speaks to the approach, but has also become more organ-saving and maximally restoring. Thirty years ago, a patient with a tumor of the leg would undergo an amputation. Today, with minimally invasive and maximally saving procedures, the leg is preserved. There has been an evolution of surgery approaches from just removing to curing without destroying.". The benefits of MIS can include small incisions, few incisions, less pain, low risk of infection, short hospital stay, quick recovery time, less scarring, reduced blood loss, etc. [3]. And MIS has become the most common method of repairing abdominal aortic aneurysms in 2003 in the United States [4]. However, traditional training method s such as that shown in Figure 1.1 can not satisfy the requirements of advanced surgical technologies developed recently.

To address these problems, efforts have taken place over the last twenty years to introduce virtual reality as part of the clinical training [5]. This is in great contrast to other industries that have fully enjoyed the benefits of computer simulations. Take for instance pilots training, that has heavily relied on flight simulators for many decades. Of course, the simulations introduced in the medical field have been intended to enhance, not replace the traditional methods but even so, they are yet to penetrate the main stream medical community. Note that among medical simulations, the modeling of tissues and organs (in particular heart and brain) has received more attention than the modeling of surgical instruments and devices. And most of these simulations are only visually correct.

So we want to develop a user-friendly, physically based surgery simulation software that provide an alternative to traditional surgical training methods. As shown in Figure 1.4, the software is composed of three major parts, user interface, simulator engine, and haptic device, which provide interface between user and software, simulate the suture according to operations by the user, and provide force feedback

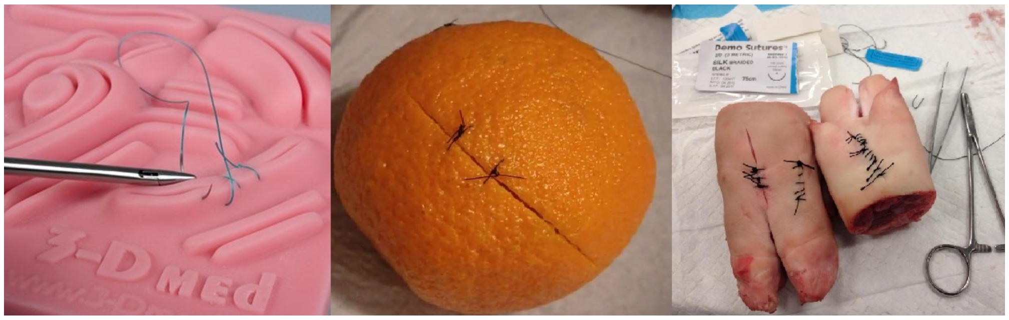


Figure 1.1: Traditional surgical training methods that include kit with plastic foam, fruit such as orange, and practicing on pig's foot.

Figure 1.2: Robotic suturing [6]

respectively. The simulator engine can be further consist of three parts which are used to simulate the dynamic motion of guide wire used in minimally invasive surgery, surgical thread, and tissue material at real time rate.

We should notice that during the process of minimally invasive surgery, the guidewire moves very slowly [8], and thus we assume the simulation of guidewire is a static problem. Since the guidewire is mainly subject to end force load [8] and we focus on the effects of the constraints on the configurations of a buckled beam (i.e.

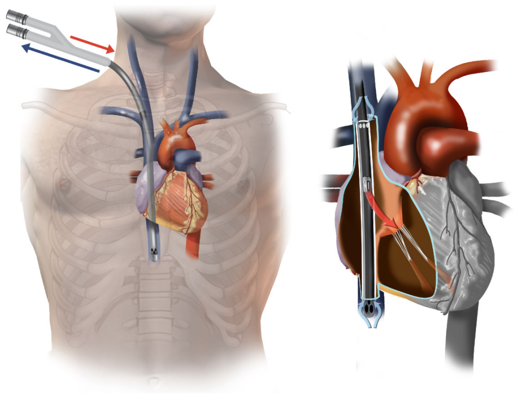


Figure 1.3: Illustration of a guide wire inserting into blood vessel [7].

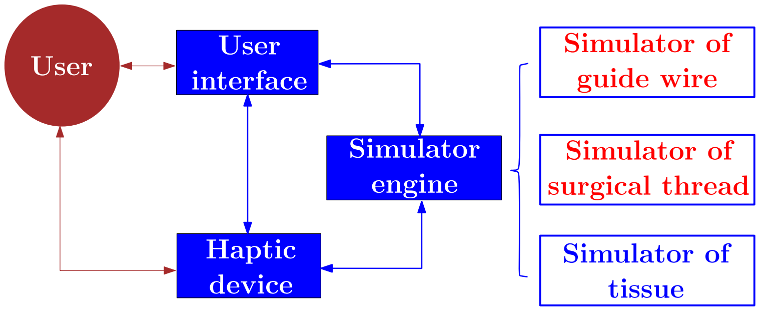


Figure 1.4: The structure of our surgery simulation software.

the deformation of tubes passing through the esophagus or the like-organs during endoscopic operations shown in figure 1.5), we consider a simplified buckling problem of a two-dimensional (2D) beam constraint between two parallel curved walls as shown in Figure 3.1. Comparing with the stiffness of the human tissue material, the guide wire is very soft [8], and therefore the constrained channel is assumed to be solid. Furthermore, the friction between the beam and solid curved walls is neglected due to the low friction between guidewire and the tissue material of human body. So the simulation of guidewire is assume to be a problem of static buckling of Euler beam constrained by solid frictionless curved channels.

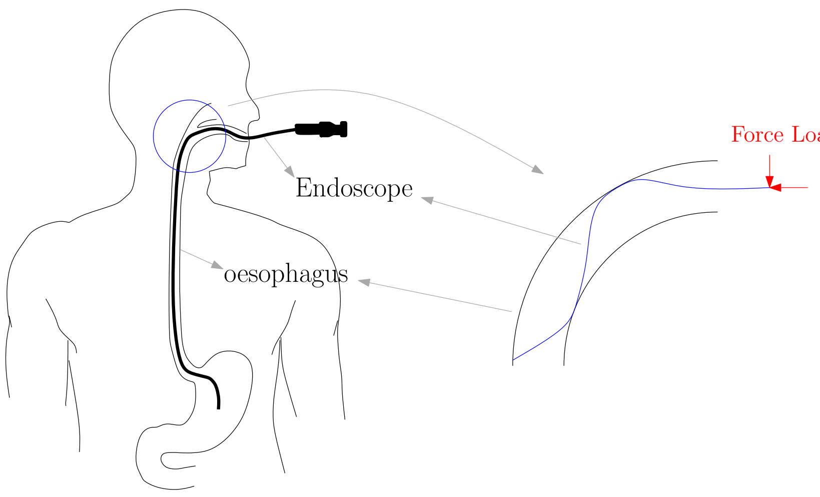


Figure 1.5: Right: The figure shows the endoscope passing through oesophagus. Left: the planar beam buckling problem we are interested.

For the simulator of surgical thread, the surgical thread should be stiff enough to hold steadily the knot avoiding slacking, so the surgical thread is often considered to be either an inextensible or a very stiff rod. The research on surgical thread with high stiffness have been done by Hüsken [9]. Thus we will focus on the simulation of inextensible surgical thread in this dissertation.

The real-time simulation of rope, thread or rod, and knot tying in particular (see Figure 1.6), raises a lot of difficult issues. A lot of research works have been done in this area. One of the major challenges to simulate the thread is its physical model. Despite the fact that the configuration space of the rod has only one dimension, its mathematical representation is difficult. This comes from the observation that an elastic rod cannot only bend, but also twist around its centerline. Thus, the configuration of a deformed rod cannot be described in terms of the position of its centerline alone. Instead, the orientation of each cross-section introduces an additional degree of freedom. Since the positions and orientations are mechanically coupled and the inextensibility of the rod should also be considered, the rod is an intrinsically constrained system. Additionally, when tying a knot, the thread will collide itself, so management of self-collisions should be take into consideration during the simulation. When two segments of the thread are close to each other and self-collision is going to happen in continuous time, we should notice that in discrete time of the numerical simulation, the diameter of the thread is so small that it may result in the situation that the two segments are separated without collision at a time step; while at the next time step, the two segments of the thread may cross each other without detect the self-collision actually happed in continuous time. This is because the relative distance between the two segments changed during the current time step is larger than the original distance between these two segments in previous time step. This tunnel effect raise the challenges in the collision detection and response. Moreover, surgery always takes a long time, which requires long time simulation. Particularly, it may take more than 10 minutes for a novice student to tying one knot, which results in another challenge, the stability of long time simulation. Challenges associated with the real time simulation of deformable objects in a three-dimensional space have been recognized by the computer animation industry [10].

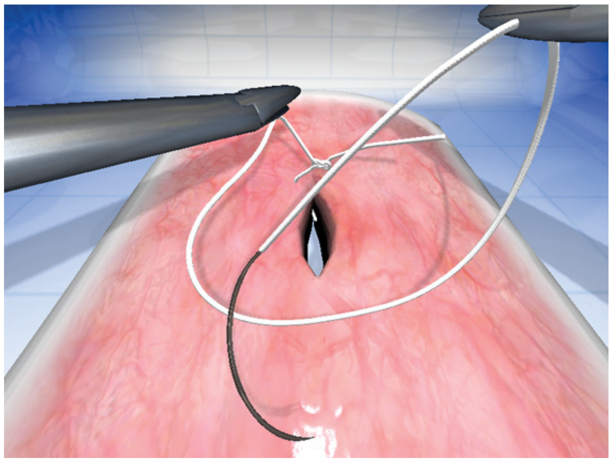


Figure 1.6: Surgical thread knot tying [11].

1.2 Objectives

The objective of this dissertation is to develop novels techniques for modeling the static buckling of Euler beam with circular channel constraints subject to end forces load which is core model of the guidewire simulator, and the dynamic response of inextensible elastic Kirchhoff rods which can be served as a surgical thread simulator of the user-friendly, physically based surgery simulation software as shown in Figure 1.4.

In the following part of the dissertation, we first discuss the related works on the simulation of beam and rod in the past; then we develop the models for static simulation of Euler beam in two dimensional space and the dynamic simulation of Kirchhoff rod in three dimensional space. Additionally, we attach the simulation codes for these two models in the appendix.

2. RELATED WORK

Various kinds of techniques have been developed to simulate the static and dynamic response of beam or rod in the past. In this section, we will first discuss the research on the static buckling of constrained Euler beam and then review the dynamic response of elastic rod.

2.1 Static simulation of a buckled Euler beam with constraints

Beams with various boundary conditions have been reported in the literature. In particular, when we consider laterally constrained beams, Feodosyev, V.I. [12] investigated the buckling configuration of a beam constrained by a narrow straight channel. The critical forces and the corresponding stable deformation patterns are analyzed with the assumption of small deflection in this work.

Domokos et al. [13] have researched on the buckling configuration of a beam with large deflection constrained by a frictionless wide straight channel (see Figure 2.1). The procedure of their work is first conduct experiment and observe the deformation patterns; then all deformation patterns are treated independently and the corresponding governing equations are developed and cataloged; lastly the final solution is obtained through combing all these independent governing equation together. Chai [14] has shown the buckling configurations of a simply supported beam with the constraints of parallel straight solid walls by assuming point/line contact regions and computing the reaction forces in these regions to find the buckled shape. Later Holmes et al. [15] consider elastic buckling of an inextensible beam with hinged ends and fixed end displacements, confined to the plane, and in the presence of rigid, frictionless sidewalls which constrain overall lateral displacements. They formulate the geometrically nonlinear (Euler) problem and develop global search and path-following algorithms to find equilibrium in various classes satisfying different contact patterns and hence boundary conditions. They derive complete analytical results for the case of line contacts with the sidewalls, and partial results for point contact and mixed cases.

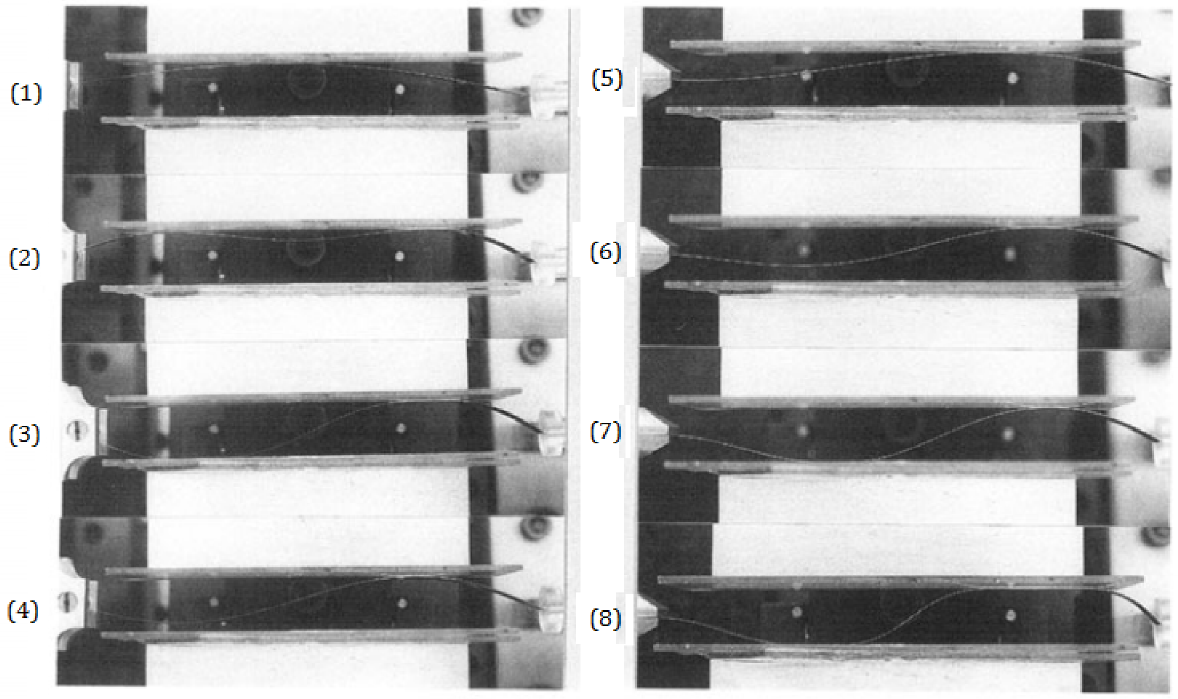


Figure 2.1: Some typical buckled shapes research in the work by Domokos et al. [13]. Left column: pinned-pinned beam; right column: clamped pinned beam.

Chen et al. [16, 17] has investigated the 2D beam buckling problem of in a channel with symmetric boundary conditions, as shown in Figure 2.2. In their work, both the inside and outside channel walls were assumed to be circular. They first carried out a sequence of experiments and then identified different deformation patterns of the beam, i.e., segments of beams in contact with the channel walls. Based on the observed patterns, each segment of the beam was listed independently, and the corresponding differential equations together with boundary conditions were developed. The full solution resulted in combining all solutions associated with the different patterns. The technique by Chen et al. has the advantage to be straight forward and easy to solve 2D beam buckling problems in a circular channel with symmetric boundary conditions. However, the approach requires experimentations for each geometry and boundary conditions to identify deformation patterns, which can then be incorporated into the segments of the differential equation. This is because the solution method depends upon knowing the type of contacting and non-contacting regions in priori. This severely limits the application of the approach since each geometry and boundary conditions have to be reinvestigated.

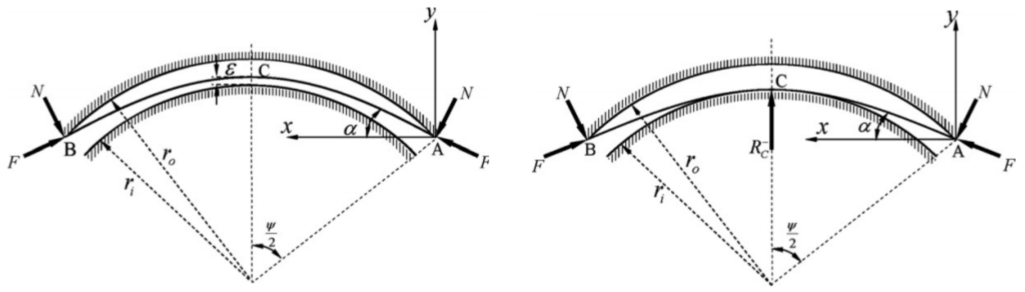


Figure 2.2: Two buckled patterns in the work by Chen et al. [16]. Left: the middle point of the deformed wire is elevated from the inner radius a small distance ε. Right: the deformed wire is in point contact with the inner radius of the tube at the middle point C.

Later, Narayanan et al. [18, 19] develop an innovative model to find the configuration of a buckled Euler beam with straight channel constraints based on energy minimization method, which allows to find regions of contact without the need to resort to experiments or assuming deformation patterns priori the simulation. In

Narayanan's work, the beam is discretized in both domain and range, and the discrete minimization of an energy functional has a special Markov structure that can be exploited by means of the well known Viterbi algorithm to find the minimum. The core idea of this method is that, a discrete set of finite admissible configurations of the beam is first constructed, and then the Viterbi algorithm is employed to search one configuration of the beam with minimum potential energy among all the discrete admissible configurations constructed. Our approach to simulate the buckled Euler beam with circular channel constraints is based on this model, and the advantages of this method, without the need to resort to experiments or assuming deformation patterns priori the simulation, has also been kept. The details of our work is shown in Section 3.

2.2 Dynamic simulation of elastic rods

The limitation of computational power of early years meant that physical accuracy had to be sacrificed to generate visually plausible simulations quickly. So we will first discuss the purely geometry-based simulation techniques wherein the models focus only on visual accuracy and not force feedback have been developed.

2.2.1 Purely geometry-based simulations

Many early efforts in modeling surgical thread are devoted on the model of **spline** [20]. Generally, the curves are represented by a set of control points. These control points are used to weight a linear sum of basis functions associated with each spline type. The parametric curve $C(u)$ is defined by

$$C(u) = \sum_{i=0}^{n} P_i B_i(u) \tag{2.1}$$

where P_i are the control points and $B_i(u)$ are the basis functions, see Figure 2.3. The designer adjusts the shape of the curve by moving control points to new positions, by adding or deleting control points, or by changing their weights. Terzopoulos et al. [21] simulate elastically deformable objects by employing spline model. While they focus on a general model for elastically deformable objects, one of the examples they mention is a telephone cord.

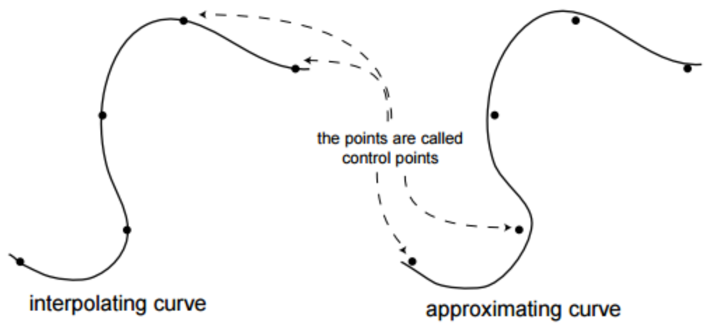


Figure 2.3: Two types of spline model: interpolating curve (passing all the control points); approximating curve (not necessarily passing all the control points) [20].

Phillips et al. [22] apply the model of splines to simulate tying knot, as shown in Figure 2.4. In their work, the rope is modeled as a spline of linear stretching springs, with spheres placed on the control points to represent the rope volume. The spheres tend to stretch apart or bunch up and thus insert or delete the control point of the curve respectively, which make the control points adaptive with respect to the inter-points distance during the simulation. The collision handling based on an impulse model allows to preserve the topological properties of the thread. However, this system does not operate in real time, as it takes several minutes of computation to simulate the gravity effect of rope with two end pinned to a fixed point. Moreover, the bending and twisting properties of the thread are not considered.

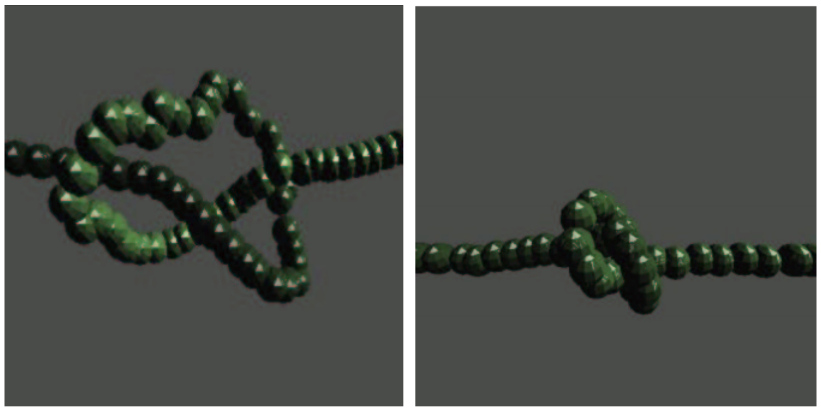


Figure 2.4: Tying a square knot based on the model of spline in the work by Phillips et al. [22]. The rope is modeled as a spline of linear stretching springs, with spheres placed on the control points to represent the rope volume. The bending and twisting properties of the thread are not considered.

Lenoir et al. [23, 24] create a real-time spline based simulator, in which the stretching and bending behavior are achieved through the springs distributed over the spline representing the surgical thread (see Figure 2.5). They are also able to simulate knots by handling self-collisions with penalty methods. However, torsion is not included.

To simulate the torsional behavior, Theetten et al. [25] introduce an additional degree of freedom, the rotation field, for each control point of the spline, see Figure 2.6. The rod centerline positions $\mathbf{r} = (\mathbf{x}, \mathbf{y}, \mathbf{z})$, and a rotation field θ representing the orientation of the cross-section. They then express the rod through a set of polynomial spline curves: $\mathbf{q} = (\mathbf{r}, \theta) = (\mathbf{x}, \mathbf{y}, \mathbf{z}, \theta)$; and each resulting spline is expressed as $\mathbf{q}(u) = \sum_{i=1}^{n} b_i(u)\mathbf{q}_i$, where $b_i(u)$ is the the i_{th} spline function for the control points

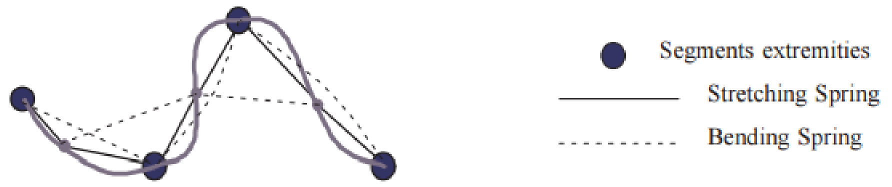


Figure 2.5: Spring distribution over a spline in the work by Lenoir et al. [24]. The stretching and bending behavior are achieved through the stretching springs and bending springs. Twisting are not considered.

$\mathbf{q}_i$. The model is then developed from the Lagrangian equation, in which the potential energy and kinetic energy of the system are derived based on both the positional and torsional degree of freedom.

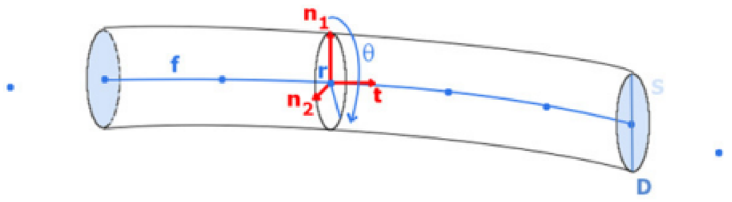


Figure 2.6: Scheme of a rod with its geometrical parameters and local frame in the work by Theetten et al. [25]. Torsion are considered through the introduction of rotation field θ.

Brown et al. [26] develop a simulator based on the model of **Follow the Leader** (FTL) that firstly enables realistic interactive simulations of complex knot tying operations at real-time rates. In their work, The thread is discretized into N nodes. When one control node is grasped, the motion occurs and the rest of the thread is propagated to follow the control node according to the algorithm of FTL, such that the inextensibility of the thread and some of the most important properties of tying knots (namely, that objects do not pass through or penetrate each other but rather slide with some friction) are maintained (see Figure 2.7). This approach is computationally efficient and allows hard constraining of the total length of the thread. However, it is also very limited. Bending and torsion, which can be essential for correct thread behavior in a suturing situation, are not supported. In general due to its non-physical nature, it is difficult to model force based effects such as gravity. Later, Müller et al. [27] extend the algorithm FTL to Dynamic Follow-The-Leader (DTFL) for hair simulation, allowing a more dynamic behavior of the hair strands.

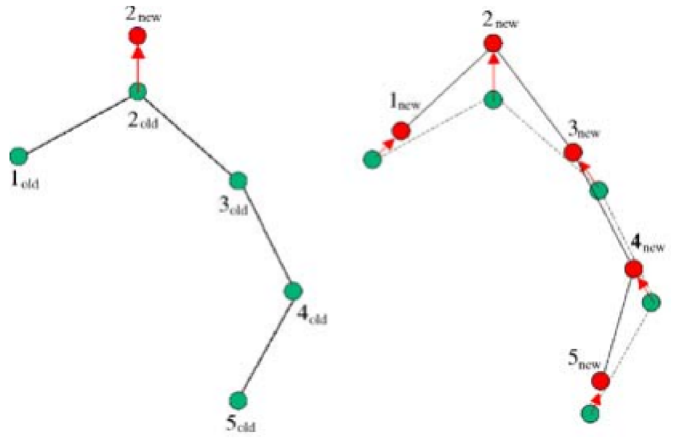


Figure 2.7: If we move node 2 (left), nodes 1 and 3 follow, then 4, then 5 (in the work by Theetten et al. [25]).

One of the most successful purely geometry-based model is **Position-based dynamics** (PBD) developed by Müller et al. [28] to efficiently and robustly simulate clothing. PBD has primarily been used to simulate various physical phenomena associated with solid and thin-shell deformations in many game engines and visual effects, where speed and controllability is crucial. Umetani et al. [29] extend the application of PBD to rod like one dimensional object. PBD requires an object's deformations be characterized with discrete positions of points. However twist of a rod cannot be directly specified with only positions of the rod center line, because twist requires angular information - rather than positional - which describes how much the material is twisted around the centerline. Ghost points are introduced and placed on edges to represents material points distributed around the edge, and the orientation of material frame naturally follows the rods' tangential direction, leading to an efficient formulation. However, due to the adoption of Rodrigues' rotation to represent the material frame, the relative rotation angle for each two connective segments is limit to π.

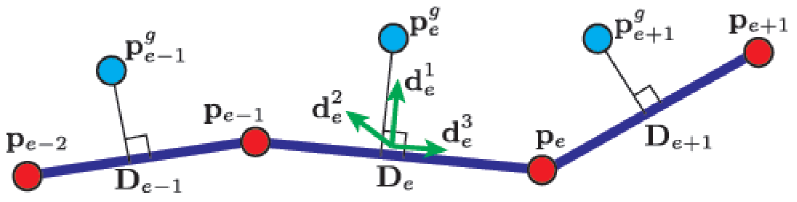


Figure 2.8: Configuration of edges, nodes points (red) and ghost points (cyan) (in the work by Umetani et al. [29]). The discrete centerline is expressed through the nodes points and the orientation is represented by the ghost points.

With purely geometry-based methods, the systems require the prior knowledge of the objects being manipulated and deformations must be explicitly specified, since no constitutive relation or balance law is used to predict the motion, which indicate that modeling is dependent on the expertise of the user. This has prompted the graphics community to begin exploring physically-based methods for animation as computing power and graphics capabilities boost during recent decades. These methods rely on physical principles and computational power to simulate complex processes that would be difficult or impossible to model with purely geometric techniques [30].

2.2.2 Physically based simulations

Mass-spring systems are one physically based technique that has been used widely and effectively for modeling deformable objects. Mass-Spring-Damper models are dynamic models. The spring forces acting on each node, $\mathbf{F_s}$, are usually considered to be linear (Hookean's Law). This means that the spring force has a magnitude directly proportional to the displacement of the spring from its rest position and acts to restore the spring to its initial position.

The earliest mass spring model for rod-like one dimensional objects known to the author of this thesis is introduced by Rosenblum et al. [31]. To simulate large quantities of hair, they employ a very simple model of mass points connected by springs. Additional hinge springs provide bending resistance, as shown in the Figure 2.9.

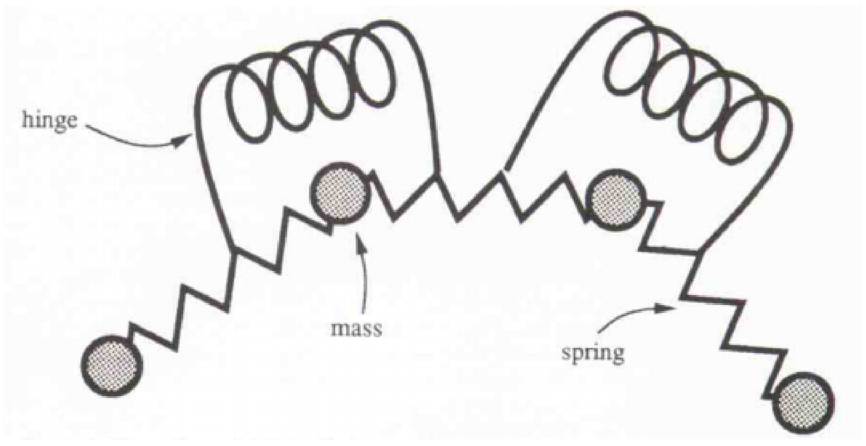


Figure 2.9: Dynamic model for a single strand of hair in the work by Rosenblum et al. [31]). The hinge springs provide bending resistance.

Several techniques have been developed to improve the mass spring system to simulate the torsional behavior. In the work by Wang et al. [32, 33], the bending and torsion behaviors are considered by introducing angular spring and torsion spring and measured according to the relation among the adjacent points.

Choe et al. [34] develop a new hybrid model based on mass spring system to simulate hair. They treat hair as a collection of wisps, which are discretized into several segments. The segments of a wisp are represented as rigid bodies with a center of mass and connected as a serial chain by linear and angular springs. The orientation of the segment is represented by quaternions, which is employed to measure the bending and twisting of the hair.

To simulate torsion, Selle et al. [35] introduce torsion spring and tetrahedral altitude springs (to prevent the collapse of the hair model where all the springs and masses are in the plane) to the mass spring system, see Figure 2.10. For curly hair (left), the altitude springs are connecting each discrete nodes and the one three nodes away from it; for straight hair, perturbed points are introduced so that the altitude springs can still be applied to represent the twist. Also, the inextensibility of the hair is achieved by strain limiting approaches[36][37].

Kubiak et al. [38] have improved the mass-spring model to simulate the bending and twisting effect of thread by adding bending springs and torsion parameter 2.11. The bending stiffness is based on the distance between two different ends of two continuous segments. The torsion is evaluated by torsion angle, the angle difference between the material frame and the Bishop frame [39], which is the same as our model to measure torsion and the details is shown in our part.

Mass-spring systems are intuitive, generally easy to implement, and are computationally efficient, making real time animations possible. Such systems handle even large topological deformations with relative ease. They are also well suited to

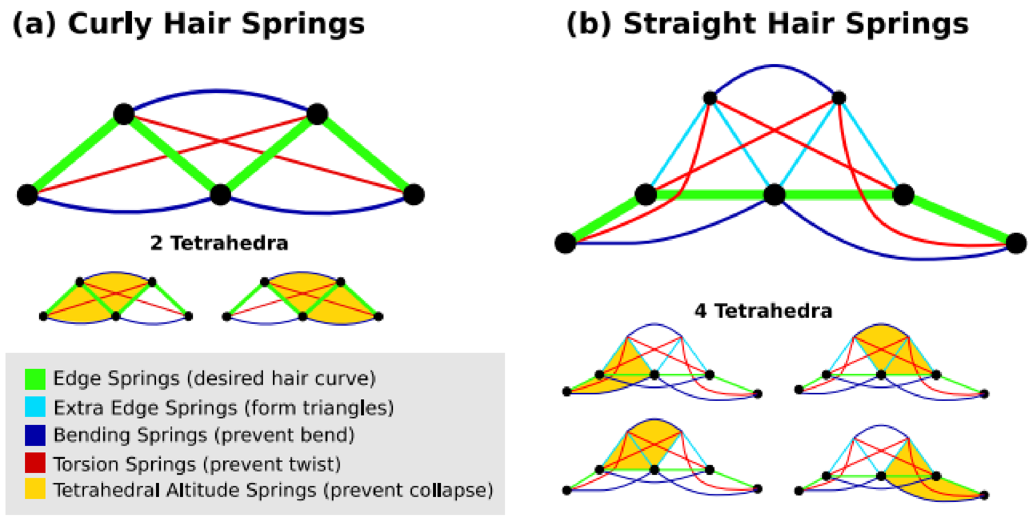

(a) Curly Hair Springs
(b) Straight Hair Springs
2 Tetrahedra
4 Tetrahedra
🟩 Edge Springs (desired hair curve)
🟦 Extra Edge Springs (form triangles)
🟦 Bending Springs (prevent bend)
🟥 Torsion Springs (prevent twist)
🟨 Tetrahedral Altitude Springs (prevent collapse)

Figure 2.10: The mass spring model of hair in the work by Selle et al. [35].

parallel computation due to the local nature of interactions. The main drawback associated with using Mass-spring systems is that the discrete model imposes significant approximations of the true physics that would occur in a continuous body. This approximation of elasticity theory means that it can be difficult to define the spring parameters that result in convincing simulations. Mass-spring systems also suffer from a problem known as stiffness, where large spring constants lead to poor system stability. To ensure stability, the system must be integrated over small time-steps, resulting in very slow simulations. On a more general note, the local nature of interactions means that for global deformations, there is a delay in the propagation of force effects through a given system. Finally, Mass-spring systems have a tendency to oscillate due to their iterative basis [30] .

In the work by Anjyo et al., a simplified linear cantilever **beam** model based on Euler beam theory is employed for hairstyle modeling (see Figure 2.12), which allows hairdressing variations with volumetric and realistic appearance [40]. In order to

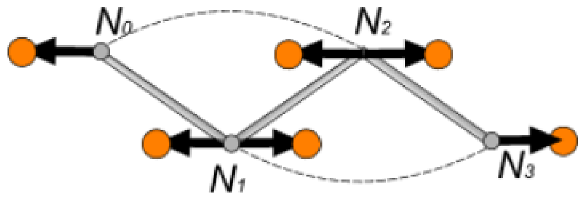


Figure 2.11: The model of thread in the work by Kubiak et al. [38]. The straight rod segments connecting two continuous nodes are the stretching spring. The springs connecting a node and the node two points away provide bending resistance. The torsion is calculated through the angle difference between the material frame and the Bishop frame [39].

describe the dynamical behavior of hair in an animation, one-dimensional projective differential equations of angular momenta for linked rigid sticks are also derived. The pliability of hair can be controlled by using a set of stiffness parameters in the method. The same deformation model has been used by Daldegan et al. [41] to develop a framework for the simulation of hair. However, the nonlinear behavior of large deformed hair is modeled based on linear Euler beam theory. Moreover, the twist effect is still not considered.

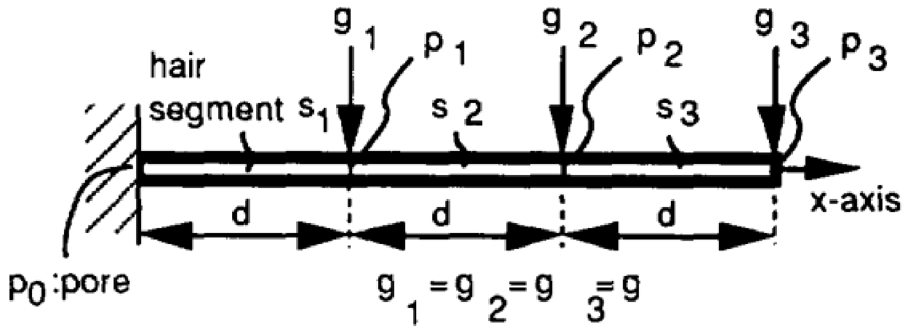


Figure 2.12: The model of hair based on the theories of Euler beam in the work by Anjyo et al. [40].

Lenoir et al. [42] developed a simulator based on finite beam element model to simulate extensible guidewire. The model is defined as a set of beam elements. Each element has 6 degrees of freedom, 3 degrees of freedom in translation and 3 in rotation, which allow that the elements can model bending, twist and other deformations. By avoiding resolving the global stiffness matrix, it reduced the computational time significantly and made real-time interaction possible.

A common starting point for physically based simulation of rods or threads which includes torsion behavior is the **Cosserat rod** [43]. The core idea of the model is that the centerline of the rod is represented by a curve, and the orientation of the rod is described through material frame, which is Seret-Frenet frames assigned to each point of the curve 2.13. Pai [44] have developed a model only allow for simulating the static deformation of a strand and introduced the Cosserat theory in the fields of computer graphics. Bertails et al. [45] provide a robust solution for simulating the dynamics of hair strand based on theory of Cosserat rod. Their approach is able to simulate natural curly hair strands. The advantage of the model based on Cosserat rod theory is the consistently handling of bending and torsional effects. However, the implicit representation of the centerline complicates the handling of collisions.

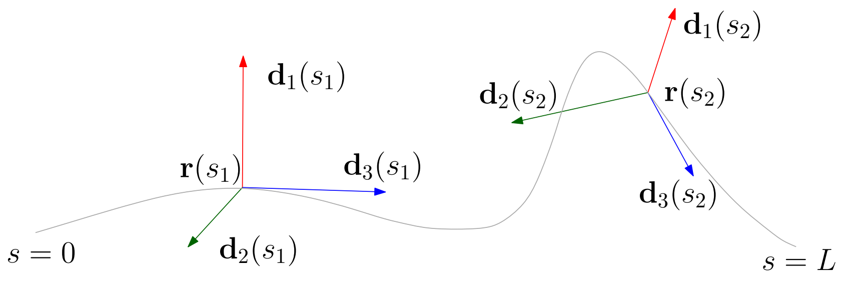


Figure 2.13: Representation of Cosserat rod [46]. The curve represent the centerline of the rod. The material frame $[\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3]$ represent the orientation of the rod.

An improvement over Pai's model is contained in the work of Spillmann et al. [46]. They developed a deformation model called 'CoRdE' to address the contact handling problems most Cosserat models have due to the lack of an explicit representation for the center line. They express the rod centerline using an explicit scheme and introduce additional constraints to relate the implicit (orientation of rod cross section) and explicit schemes (rod centerline). They have discretized the rod into elements, and derived the Lagrange equations for each element. By numerically combining all these equations, the rod is evolved in time. In their later work [47], they extend their approach with a method for adaptively subdividing the rod depending on the local curvature, allowing them to build knots with short segments while not reducing performance with a high number of vertices. Punak et al. [48] have optimized the performance of the 'CoRdE' model by neglecting internal friction and other dissipate effect as well as kinetic effects of the rod. These simplifications result in an increase of computational speed while keeping the model relatively accurate.

Bergou et al. [49] developed a model for a discrete elastic rod with an explicit scheme for the rod centerline based on the **Kirchhoff theory of elastic rods** [50, 51] to simulate inextensible thread like object. Torsion is measured by the difference of angles between the material frame and the Bishop frame of the rod [39] so that no additional degree of freedom needs to be introduced to represent the torsional behavior. In addition, torsion is assumed to propagate instantaneously because the rod has a very small polar moment of inertia and this reduces the computation time needed to capture the temporal evolution of torsional waves. Representing the centerline with an explicit scheme and handling the torsional behavior easily make this model computationally-efficient while physical accuracy is respected. However, the inextensibility of the thread is achieved through a manifold projection method, which is not a physical based techniques. Compared with models based on Cosserat rod theory, this model is extremely suitable to simulate the dynamic behavior of one dimensional rod like object due to explicit representation of centerline, low degree of freedom (only four) but still keeping the benefit of models based on Cosserat rod theory-naturally handling bending and torsion.

Maisheng Luo et al. [49] have been inspired by the model of discrete elastic rod and develop a model to real-time simulate the guidewire of vascular intervention based on Kirchhoff rod [52]. In their work, the long slender bodies of guidewire is simulated by using more efficient special case of naturally straight, isotropic Kirchhoff rods. The inextensible constraint of the guidewire is achieved by fast projection algorithm [53]. They derive the equations of motion for guidewire with continuous elastic energy and discretize these equations using a linear implicit scheme that guarantees stability and robustness. In addition, the system is further augmented by the force sensing device as input. Other similar noteworthy articles report on modeling discrete elastic rods based on the work by Bergou et al. has also been done by Tang et al. [54, 55] and Huang et al. [56].

Nathan Hüsken [9] have also been inspired by Bergou et al.'s work [49] and develop a model to real time simulate the behavior of stiff surgical thread. In his work, the surgical thread is assumed to be very stiff, and implicit integrator is employed to reduce the instability caused by the stiff springs. Collision are managed through constraints.

There also exists many other schemes to model the dynamics of thin rods. For example, Grégoire and Schömer [57] propose a novel hybrid model that combines mass-spring system and Cosserat rod. In it, the rod is modeled as a mass-spring system but bending and torsion follow the Cosserat theory. From the above review, we find that the discrete elastic model developed by Bergou et al. [49] based on the theories of Kirchhoff rod is one of the best physical based approach to simulate the rod like one dimensional object in three dimensional space. since the model is naturally handling the bending as well as twisting, and its degree of freedom are only four (three for position, and one for torsion). Our work for the dynamic simulation of rod is also based on this model, the details are shown in Section 4.

3. SIMULATION OF STATIC BUCKLING OF EULER BEAM SUBJECT TO ONE SIDED CONSTRAINTS*

To study the configurations of a buckled beam with constraints, we will first conduct qualitative experiments.

3.1 Qualitative experimental observations

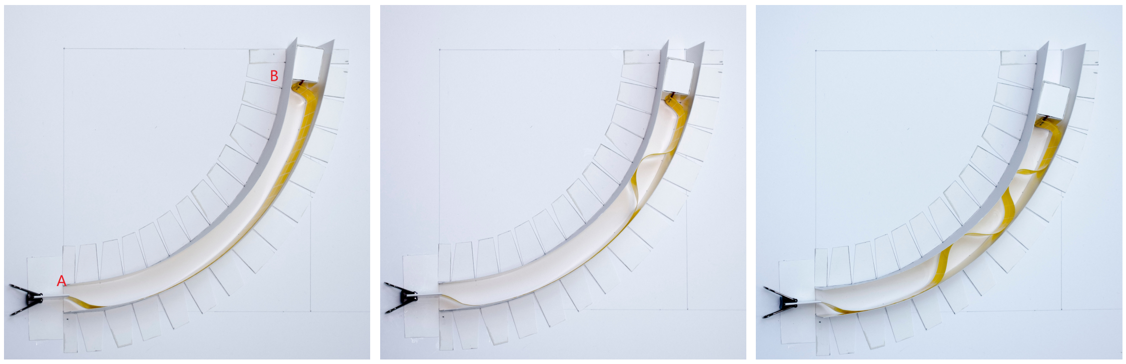
Qualitative experiments have been used by Chen et al. [16, 17] to visualize the possible buckled shapes of a beam with symmetric boundary conditions (i.e. the two ends of the beam are either free to move or clamped) and constrained by a circular channel.

In contrast, this work is broader in the sense that we seek to study problems with non-symmetric boundary conditions such as those with one end clamped and one end pined. Thus the experimental results by Chen et al. can not be used to serve as direct comparison with our simulations, which leads us to conduct our own experiments (see figure 3.1) to be used as benchmark.

In order to illustrate the technique we developed in this dissertation, we first consider the case of a circular channel (we stress that the technique is not limited to circular channels. The generalization to non-circular channels is shown in the results of this section). Figure 3.1 depicts the set-up. Poster-board type paper is bent to form a quarter circular channel with outer radius $R_o = 20cm$ and inner radius $R_i = 18cm$. A plastic strip initially straight is inserted into the channel to simulate the planar deformation of the beam. Referring to Figure 3.1, the boundary

*Reprinted with permission from "A direct minimization technique for finding minimum energy configurations for beam buckling and post-buckling problems with constraints" by Zhujiang Wang, Annie Ruimi, A.R. Srinivasa, 2015. International Journal of Solids and Structures, Volume 72, Pages 165-173, Copyright 2015 by Elsevier Ltd.

conditions are that the left end (A) is clamped and that the right end (B) is pinned to a cube made of paper free to move along the channel, and thus a compressive load is applied at B and subsequent buckling shapes are observed. As the load is increased, the corresponding progressive stable configurations of the strip (in equilibrium state) are shown from Figure 3.1a to 3.1c. These qualitative observations are compared to our simulation results (see Figure 3.17).



(a) Pattern 1 (b) Pattern 2 (c) Pattern 3

Figure 3.1: Experimental observations of deformation of an elastic strip in a quarter circular channel. The left end of the beam is clamped and the the right is pinned to a small piece of paper compressed into a cube. Note that the progressive appearance of buckled patterns are compatible with simulations in the figure on p. 47.

3.2 Simulation

Traditional methods used to solve the buckling problem of beams relies on the solution of the differential equation or on finding the minimum potential energy [58]. For the latter methods, the resulting minimization problem is very challenging to solve when the beam is geometrically constrained in a curved channel. In this work, we will solve the problem by employing a new technique based on a Dynamic Programming Approach (Bellman, 1954) [59] called the Viterbi algorithm [60]. While this algorithm is well known in the information theory literatures. The fact that it can be used for the solution of constrained beam buckling problems was already noted in the work by Narayanan et al. [18, 19]. For the cantilever beam problem, the differential equation together with the initial conditions has a special Markov structure that can be gainfully exploited. The Markov structure makes the dynamic programming approach suitable for this problem and allow us to sequentially optimize it. To better understand the how Viterbi algorithm works on beam buckling problem, we will first introduce Viterbi algorithm through a simple path optimization example and then make a brief introduction to the work by Narayanan et al. [18].

3.2.1 Introduction to the Viterbi algorithm

Assume that four persons want to traverse a suspension bridge at night (see Figure 3.2) [61]. The four persons A, B, C, and D, take 5, 10, 20 and 25 minutes respectively to cross the bridge. There are two constraints here: 1. the bridge can carry no more than two persons at a time; 2. each party must carry a torch while crossing the bridge. We need to find a optimization path to cross the bridge that costs minimum time.

To employ Viterbi algorithm solve this problem, we first construct the solution space of all the possible paths as shown in Figure 3.3. And the time cost for each path are shown in Figure 3.4. We take bottom path $0 \rightarrow AB \rightarrow A \rightarrow ABC \rightarrow AB \rightarrow ABCD$ as an example: 1. initial state (no body cross the bridge); 2. $A\&B$ together cross the bridge and take 10 minutes (the slower one B dominates the time cost); 3. B then takes 10 minutes to go back and A stay in the other end of bridge now (A have crossed the bridge); 4. $B\&C$ together cross the bridge which takes 20 minutes (C dominates this action) and A, B, C reach the other end of bridge; 5. C takes 20 minutes to go back and $A\&B$ stay on the other end of the bridge

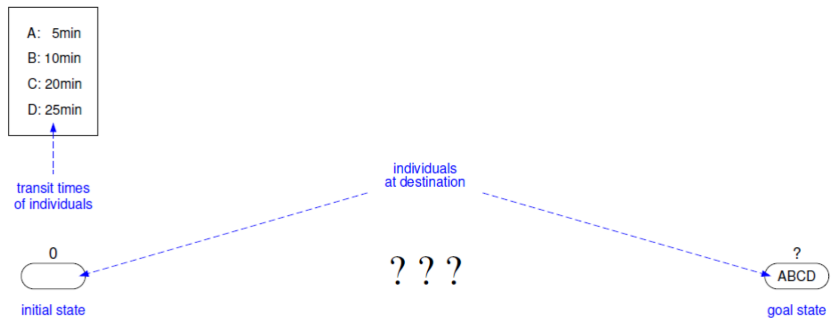


Figure 3.2: Optimization of the path problem. Four persons, A, B, C, and D, want to cross the bridge during night. They are required to carry a torch (only one) and no more than two person can cross together at a time. The initial state is empty indicating that no persons have cross the bridge; the goal state shows that all the four persons have crossed the bridge. How to choose a path that costs minimum time for the four persons to cross bridge?

now; 6. $C\&D$ then takes 25 minutes to cross the bridge and all the four persons A, B, C, D cross the bridge (reach the goal state). The total time taken by this path is $10 + 20 + 20 + 25 = 75$ minutes.

We will employ Viterbi algorithm to efficiently search the optimum path that costs minimum time from all the total $6 \times 2 \times 3 \times 3 \times 1 = 108$ possible paths (see Figure 3.5). All the optimum possible states in each stage have been searched, and the search on current stage only depends on the previous stage. For example, in the first stage, all the possible states AB, AC, AD, BC, BD, and CD are searched and the corresponding time cost are 10, 20, 25, 20, 25, and 25 minutes. In the second stage, all the possible states are A, B, C, and D. Let's take state A as an example. To reach state A, there are three possible sub-paths $AB \rightarrow A$, $AC \rightarrow A$, and $AD \rightarrow A$, and the corresponding total time cost for each path are 20, 40, 50 minutes. So we keep the sub-path $AB \rightarrow A$, which takes minimum time 20 minutes (= 10 minutes to reach AB + 10 minutes for the sub-path $AB \rightarrow A$) to reach A, and kill the paths $AC \rightarrow A$ (total $20 + 20 = 40$ minutes) as well as $AD \rightarrow A$

(total $25 + 25 = 50$ minutes), which cost more time. In this way, all the optimum paths to the other states B, C, and D in the second stage are searched, which costs minimum 15, 25, and 30 minutes respectively. In the third stage, The optimum paths to four possible states ABC, ABD, ACD, and BCD only depends on the previous stage. For example, to find the optimum paths to ABC, we only need to consider the sub-paths starting from the second stage, which are $A \rightarrow ABC$, $B \rightarrow ABC$, and $C \rightarrow ABC$, but not the whole paths starting from initial condition. We find that two sub-paths $B \rightarrow ABC$ and $C \rightarrow ABC$ cost the same minimum time 35 minutes to reach ABC, so we keep both of the two sub-path, and kill the sub-path $A \rightarrow ABC$ which takes 40 minutes. In a similar way, we search all the optimum possible state in each stage, and then find that it optimum path to reach goal state $ABCD$ takes 60 minutes. We only need to search $6+12+12+12+6 = 48$ times (smaller than the total 108 possible paths) to find the optimum path. The last step is to trace back and find the whole optimum paths as shown in Figure 3.6, where two paths exist and the minimum time take is 60 minutes.

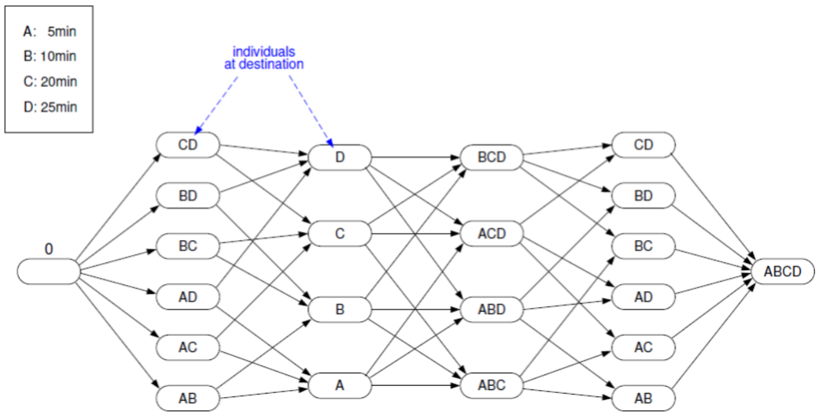


Figure 3.3: The solution space of all the possible paths. The number of total possible paths is $6 \times 2 \times 3 \times 3 \times 1 = 108$.

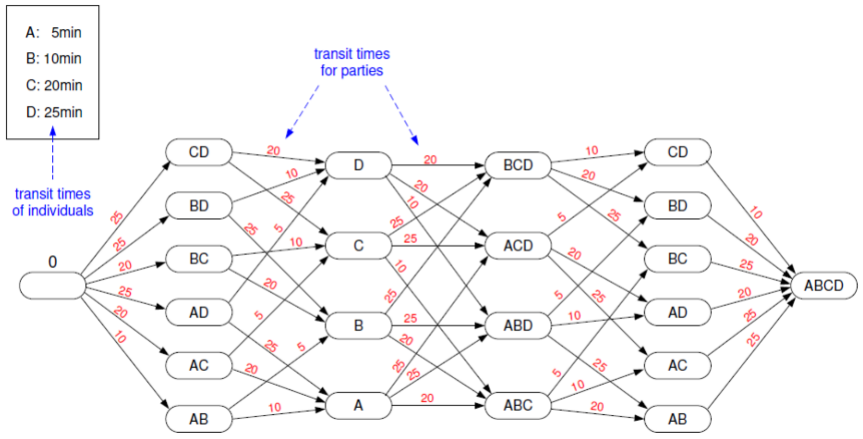


Figure 3.4: The time cost for each sub-path.

Now we will give an introduction to the application of Viterbi algorithm to find the configuration of a buckled beam constrained by straight channel in the work by Narayanan et al. [18] with a very rude or coarse discretization strategy. As shown in Figure 3.7, the problem can be expressed as,

Find $X(s)$, $Y(s)$, $\hat{\boldsymbol{\theta}}(s)$ such that

$$\begin{aligned}\hat{\theta}(s) &= arg\ \min \mathfrak{F}\left(\theta\left(s\right), T_x, T_y\right) \\ &= arg\ \min \int_0^L \left[\frac{EI_b}{2}\left(\frac{d\theta}{ds}\right)^2 - T_y \sin\theta\left(s\right) + T_x\left[1 - \cos\theta\left(s\right)\right]\right] ds\end{aligned} \tag{3.1}$$

subject to the constraints

$$-y_{limit} \leq Y(s) \leq y_{limit} \qquad \forall s \in [0, L];\ \frac{dY}{ds} = \sin\theta \text{ and } \frac{dX}{ds} = \cos\theta \tag{3.2}$$

Before introducing the application of Viterbi algorithm to beam problem, we should notice that, in the crossing bridge problem (see Figure 3.2), the goal is to

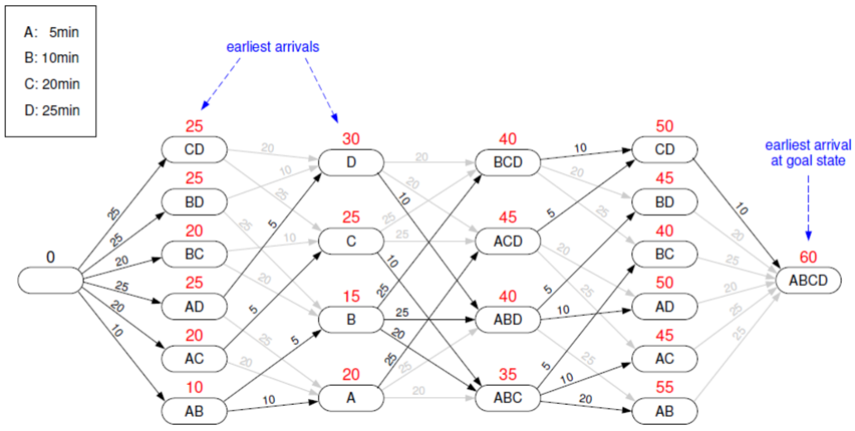


Figure 3.5: Search the optimum local path for all the stages.

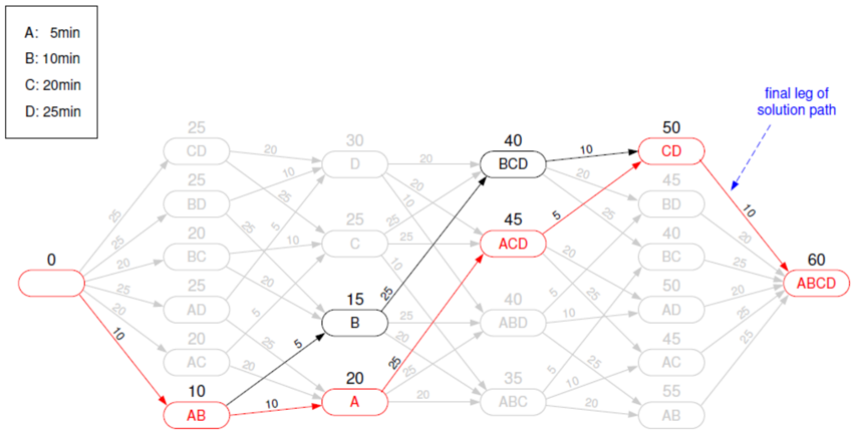


Figure 3.6: Trace back to find the optimum path that take minimum time.

optimize the path to find the minimum time cost; in the beam problem, the goal is to optimize the configuration of the beam to find the minimum potential energy. Since the crossing bridge problem is a discrete one, we should discretize (domain discretization here) the continuous problem (3.1), which is (see Figure 3.8),

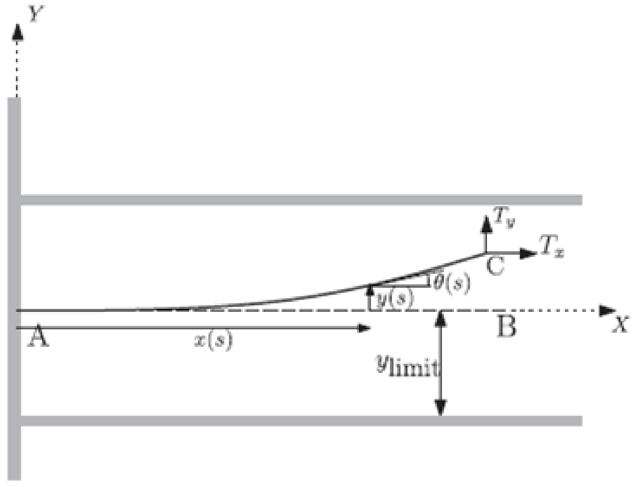


Figure 3.7: The cantilever beam is clamped at A, constrained to deform within two walls separated by a distance of $2y_{limit}$. Forces T_x and T_y act at the free end B as shown.

Find $\hat{\boldsymbol{\theta}}$ such that

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= arg \ \min \ \mathfrak{f}(\boldsymbol{\theta}, T_x, T_y) \\ &= arg \ \min \sum_{i=0}^{N-1} \left[\frac{EI_b}{2} \left(\frac{\theta_i - \theta_{i-1}}{\Delta s} \right)^2 - T_y \sin\theta_i + T_x \left[1 - \cos\theta_i\right] \right] \Delta s \end{aligned} \tag{3.3}$$

subject to

$$-\, y_{limit} \leq Y_i \leq y_{limit} \qquad i = 1, \ldots, N \qquad \text{and} \qquad \frac{Y_{i+1} - Y_i}{\Delta s} = \sin\theta_i \tag{3.4}$$

According to the application of Viterbi algorithm in the crossing bridge problem, the first step is to construct the solution space (see Figure 3.3). For the beam problem, we first discretize the range to restrict the possible positions of the beam nodes in Y direction belong to a set $\mathcal{Z} = \left\{Y^1, \ldots, Y^P\right\}$, which satisfies the channel constraints (3.4), $\max(\mathcal{Z}) \leq y_{limit}$ and $\min(\mathcal{Z}) \geq -y_{limit}$. In this way, the beam solution space is constructed, and we name the solution space as the set of discrete admissible beam

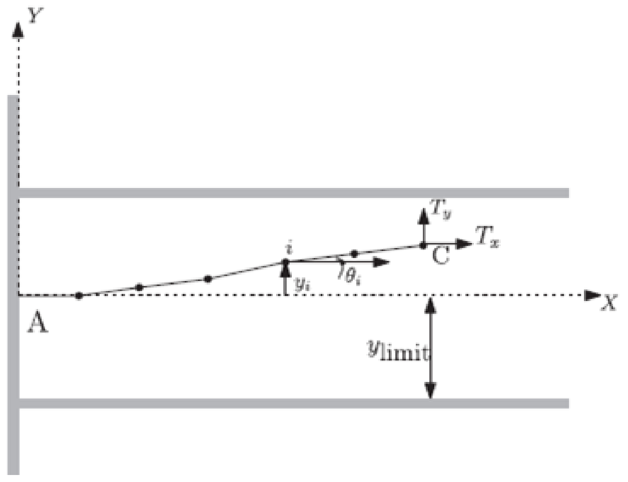


Figure 3.8: The figure shows the discretized version of the beam, with the dots representing the nodes of the discretized beam.

configurations. The beam problem with domain and range discretization can be expressed as,

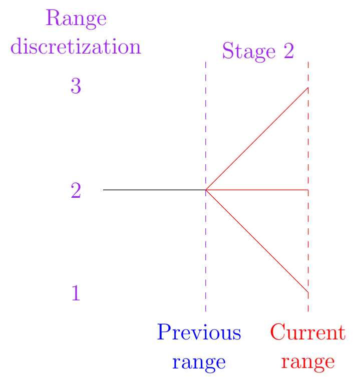
Find $\hat{\boldsymbol{\theta}}$ such that

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= arg\ \min\ \mathfrak{f}\left(\boldsymbol{\theta}, T_x, T_y\right) \\ &= arg\ \min \sum_{i=1}^{N-1} \left[\frac{EI_b}{2} \left(\frac{\theta_i - \theta_{i-1}}{\Delta s} \right)^2 - T_y \sin\theta_i + T_x \left[1 - \cos\theta_i\right] \right] \Delta s\end{aligned} \tag{3.5}$$

subject to

$$Y_i \in \mathcal{Z} = \left\{ Y^1, \ldots, Y^P \right\}, \qquad \frac{Y_{i+1} - Y_i}{\Delta s} = \sin\theta_i \tag{3.6}$$

To illustrate how Viterbi algorithm search the optimum the beam configurations, we show a simple example with a very coarse domain discretization (the beam is equally discretized into 4 elements) and range discretization (the possible vertical position can only take three values $Y_i \in \mathcal{Z} = \{-y_{limit}, 0, y_{limit}\}$). Compared to the optimum path in crossing bridge problem, a minor difference existing in beam problem is that we search the optimized sub-configurations from stage 2 but not stage 1, since the beam is clamped at A and has only one sub-configuration in first stage, as show in Figure 3.9. The symmetric beam configurations have the same potential energy (-1). We then search the possible sub-configurations in stage 3 and stage 4 as shown in Figure 3.10 and 3.11. We should notice that searching the optimum sub-configuration in each stage is the same as that in crossing bridge problem in Figure 3.5, and the potential energy in each stage only depends the previous stage.



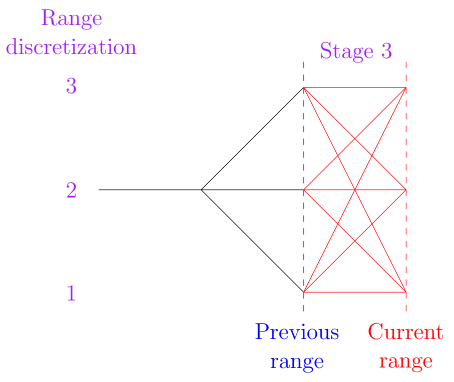
Potential energy

Previous \ Current	1	2	3
3	–	–	–
2	-1.0	0.0	-1.0
1	–	–	–

Figure 3.9: Right: the possible sub-configurations in first two stages. Left: the corresponding potential energy of the sub-configurations.

After finishing the search for all optimum sub-configuration in all stages through Viterbi algorithm, following the procedure of the path optimization in crossing bridge problem, we need to trace back to find the beam configurations (two symmetric) with the minimum potential energy (-6.0) as shown in Figure 3.12. We can choose either one of them as the final result as shown in Figure 3.13.

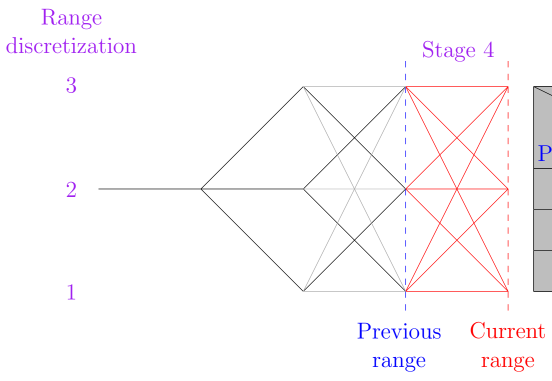
One should notice that the example illustrated here has a very crude domain and range discretization strategy. If we discretize the domain and range very fine, we can get the result shown in Figure 3.14. Up to now we finish the introduction to Viterbi



Potential energy

Current / Previous	1	2	3
3	-3.0	-4.0	-2.0
2	-1.0	0.0	-1.0
1	-2.0	-4.0	-3.0

Figure 3.10: Right: the possible sub-configurations in first tree stages. Left: the corresponding potential energy of the sub-configurations.



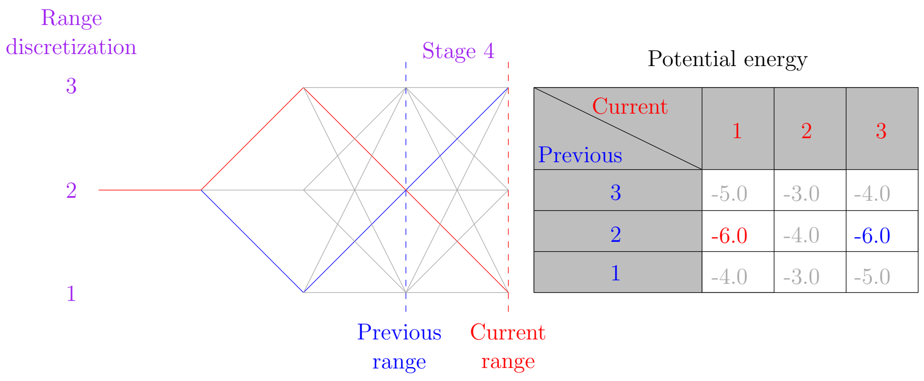
Potential energy

Current / Previous	1	2	3
3	-5.0	-3.0	-4.0
2	-6.0	-4.0	-6.0
1	-4.0	-3.0	-5.0

Figure 3.11: Right: the possible sub-configurations in all the four stages. Left: the corresponding potential energy of the sub-configurations.

algorithm.

To summarize, the key idea of the Viterbi algorithm is to efficiently search the set of possible configurations of the beam and locate the configuration with minimum potential energy. The major advantage of this technique is that, since it is based on a global search and not on a calculus based approach, it is guaranteed to find the most probable configuration among all the admissible possibilities. However, for non-linear elastic problems, there may exist many different equilibrium states for the system as long as the potential energy reaches a local minima [58]. Although we are finding the



Previous \ Current	1	2	3
3	-5.0	-3.0	-4.0
2	-6.0	-4.0	-6.0
1	-4.0	-3.0	-5.0

Figure 3.12: Two symmetric beam configuration with minimum potential energy (-6.0).

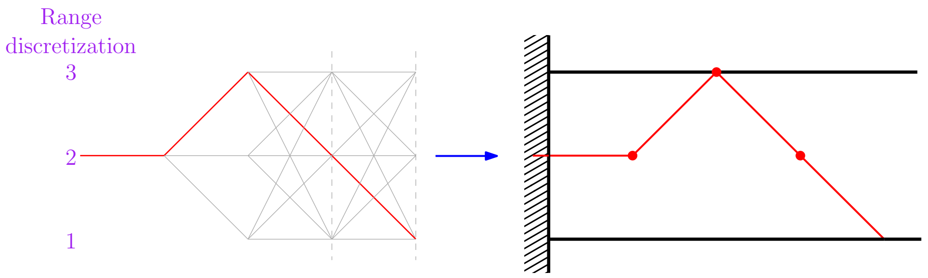


Figure 3.13: The configuration of the constrained beam.

lowest energy state among many possible configurations, we should observe that the Viterbi algorithm can be extended to find all local minima by employing a technique called list Viterbi [62]. Rather than keeping the lowest energy states, the list Viterbi algorithm can find a given number of lowest energy states[18].

Unlike calculus based methods, the Viterbi method can only search through a finite number of possible configurations. Thus the challenge here is not only to discretize the beam but also quantize the possible configurations of the discretized beam so that we arrive at a very large but finite number of possible configurations that represent the beam shape. Given these limitations, the Viterbi algorithm can be used as a means for identifying good initial configuration for subsequent gradient

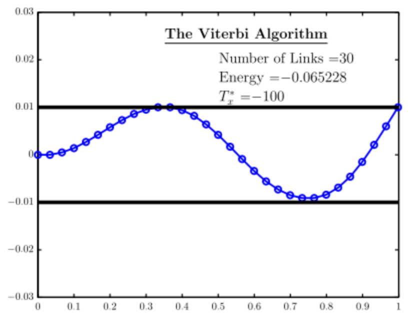


Figure 3.14: The beam configuration with a fine discretization strategy in the work by Narayanan et al. [18].

search algorithm [18].

3.2.2 Continuous problem

Figure 3.15 illustrates the geometry of the problem. we consider a beam AC that confined between the curved channel composed of walls EF and GH. The beam is assumed inextensible and frictions between the beam and the walls are neglected. The beam of length L is clamped at end A, and subject to load $\mathbf{T} = (T_x, T_y)$ at end C .

The configuration of the beam is represent by three variables $X(s)$, $Y(s)$, and $\boldsymbol{\theta}(s)$ as shown in Figure 3.15. The buckled configuration of the beam can be found by solving the following constrained energy minimization problem.

Find $X(s)$, $Y(s)$, $\hat{\boldsymbol{\theta}}(s)$ such that

$$\begin{aligned}\hat{\theta}(s) &= arg\ \min \mathfrak{F}\left(\theta\left(s\right), T_x, T_y\right)\\ &= arg\ \min \int_0^L \left[\frac{EI_b}{2}\left(\frac{d\theta}{ds}\right)^2 - T_y \sin\theta\left(s\right) + T_x\left[1-\cos\theta\left(s\right)\right]\right] ds\end{aligned} \tag{3.7}$$

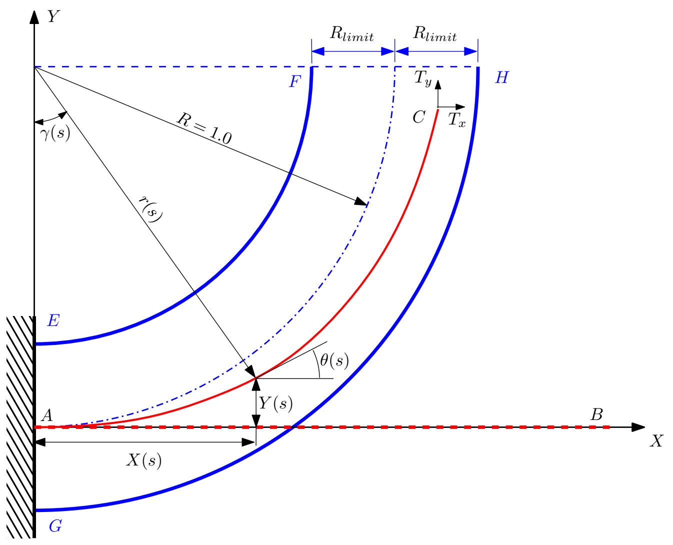


Figure 3.15: A cantilever beam AC clamped at A, constrained to deform within a quarter circular channel of radius $r_i = R - R_{limit}$ and $r_o = R + R_{limit}$ respectively. The curve line AC represents the current configuration of the beam. The load with components T_x and T_y acts at the free end C.

subject to the constraints

$$r_i \leq r\,(s) \leq r_o\;\gamma\,(s) > 0\;\forall s \in [0, L];\;\frac{dY}{ds} = \sin\theta \text{ and } \frac{dX}{ds} = \cos\theta \tag{3.8}$$

where E is the Young's Modulus; I_b is the second moment of area of the beam cross section; s is the arc length parameter along the curve; $X(s)$, $Y(s)$ are the current position of a point of the beam with respect to the global coordinate system (X, Y); $\theta(s)$ is the angle made by the tangent of the curved AC with respect to the axis X (see Figure 3.15).

3.2.3 Discretized form of the problem

According to the same scheme of Viterbi algorithm to solve beam buckling problem in the work by Doraiswamy et al. [18], we first discretize the beam problem at hand represented by eq. (3.7) with a finite difference scheme (domain discretization). We assume that, the beam is assumed to be made of N links with equal length Δs. The ends of each link are named as nodes and labeled $Node_i$, $i = 0, 1, \ldots, N$. The links are assumed to be rigid and are allowed to only rotate about the Z-axis. Let θ_i denote the angle between the X-axis and the link joining $Node_i$ and $Node_{i+1}$. Let the position of $Node_i$ be denoted by $\{X_i, Y_i\}$. The set of angles of this discretized beam is represented by an N dimensional vector $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_{N-1}]$ corresponding to the N links. Similarly, the positions of the nodes are represented by $\{\boldsymbol{X}, \boldsymbol{Y}\}$, where $\boldsymbol{X} = [X_0, X_1, \ldots, X_N]$ and $\boldsymbol{Y} = [Y_0, Y_1, \ldots, Y_N]$. Based on such a scheme, eq. (3.7) can be restated as,
Find $\hat{\boldsymbol{\theta}}$ such that

$$\begin{aligned} \hat{\boldsymbol{\theta}} &= arg \min \mathfrak{f}(\boldsymbol{\theta}, T_x, T_y) \\ &= arg \min \sum_{i=0}^{N-1} \left[\frac{EI_b}{2} \left(\frac{\theta_i - \theta_{i-1}}{\Delta s} \right)^2 - T_y \sin\theta_i + T_x \left[1 - \cos\theta_i\right] \right] \Delta s \end{aligned} \tag{3.9}$$

where

$$\frac{Y_{i+1} - Y_i}{\Delta s} = \sin\theta_i \tag{3.10}$$

subject to the constraints

$$\begin{aligned} \sqrt{(X_{i+1} - X_i)^2 + (Y_{i+1} - Y_i)^2} = \Delta s \quad i = 0, \ldots, N-1 \\ r_i \leq R_i \leq r_o \ \text{ where } R_i = \sqrt{X_i^2 + Y_i^2} \quad i = 1, \ldots, N \end{aligned} \tag{3.11}$$

where $\theta_0 = 0$ has been assumed. It is possible to eliminate θ_i by using the requirement

$\theta_i = \sin^{-1}\left(\dfrac{Y_{i+1} - Y_i}{\Delta s}\right)$. We choose to write the equations as in (3.9), because it is cumbersome to represent the discrete functional $\mathfrak{f}$ in terms of X_i and Y_i.

For conventional numerical techniques to solve the problem (3.7), usually only the domain is required to be discretized in a finite difference scheme and then solve the discretized problem (3.9), and the argument vector $\boldsymbol{X}$ or $\boldsymbol{Y}$ is allowed to take any value from $\mathbb{R}^N$ while the other argument vector is obtained from the constraints (3.11), i.e. we can get $\boldsymbol{X}$ through constraints if $\boldsymbol{Y}$ is known. While, the method based on the Viterbi algorithm also requires to discretize the range [18]. To that end, we will carefully restrict the elements of the vector $\boldsymbol{Y}$ to belong a finite subset $\mathcal{Z} = \left\{Y^1, \ldots, Y^P\right\}$, which satisfy the constrains given by Eq. (3.11). Then there would be a finite number of admissible beam configurations that satisfy the constraints constructed by the finite subset $\mathcal{Z}$.

However, we note that the algorithm in the work by Doraiswamy et al.[18] pertains to a straight channel constrains. So to accommodate the case of a curved channel constraints, we first discretize the curved channel constraints into several straight channels constraints.

Constraints discretization Figure 3.16 shows that the curved channel is discretized into S straight channels (also referred as sections). The local coordinate system $\left(x^{(k)}, y^{(k)}\right)$ is set for the k_{th} section (S_k); let the position of original point ($O^{(k)}$) of the local coordinate system be (X_o^k, Y_o^k) with respect to the global coordinate system (X, Y); let L_s^k denote the length of the section S_k and β_k denote the angle made by the S_k with respect to the X-axis, where $k = 1, 2, \ldots, S$. At this point, we can do local range discretization for each straight section, and thus we can construct a local subset of admissible beam configurations for each section (see Eq. (3.13)). Combining all the local subsets obtained, we can created a subset including a finite number of admissible beam configurations. For better illustrate the Viterbi Algorithm, domain discretization is introduced before the range discretization.

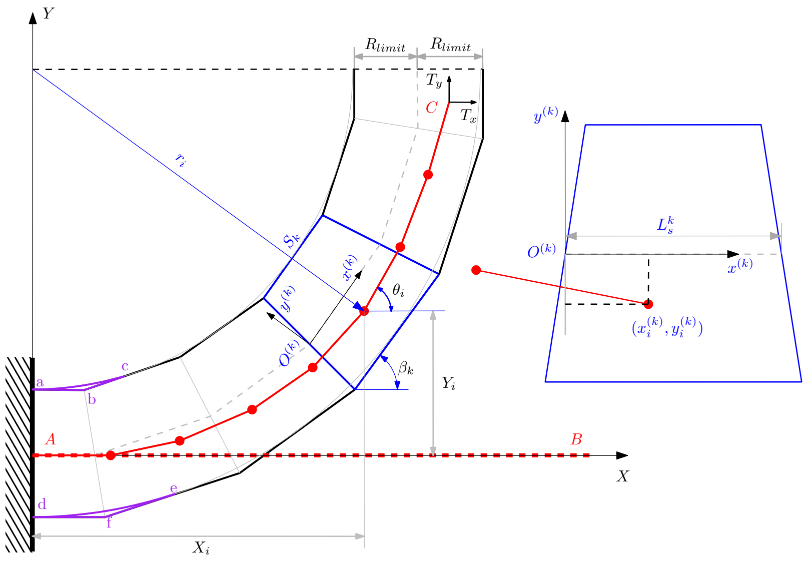


Figure 3.16: The discretized beam (the red dots in the lines AC) is constrained between several continuous straight parallel walls (the discrete curved channel constraints), and subjected to a load $\mathbf{T} = (T_x, T_y)$ at its free end C. The blue trapezoid represents the k_{th} section, and a local coordinate systems is assigned to each section.

Domain discretization in local coordinate system The domain discretization is the same as that in the work [18]. As shown in Figure 3.16, the beam itself is discretized into N equal links. The length of each link is given by

$$\Delta s = \frac{L}{N} \tag{3.12}$$

The location of $Node_i$ in S_k is referred as $\mathbf{x_i}^{(k)} = \left(x_i^{(k)}, y_i^{(k)}\right)$ with respect to the local coordinate system, where $i = 1, 2, \ldots, N$, $k = 1, 2, \ldots, S$.

Range discretization in local coordinate system In each straight section, the local range discretization is the same as that in the work by Doraiswamy et al.

[18]. As shown in Figure 3.16, we restrict the possible y-displacement of nodes in the k_{th} section with respect to its local coordinate system $\left(x^{(k)}, y^{(k)}\right)$ belong to a finite subset $\mathcal{Z}^{(k)} = \left\{y_1^{(k)}, y_2^{(k)}, \ldots, y_M^{(k)}\right\}$, where max $\left(\mathbf{y}^{(k)}\right) = R_{limit}$, min $\left(\mathbf{y}^{(k)}\right) = -R_{limit}$. In this paper, we choose

$$y_m^{(k)} = -R_{limit} + (m-1)\Delta y, \text{ where } \Delta y = \frac{2R_{limit}}{M}, m = 1, 2, \ldots, M, \;\; M \in \mathbb{N}^+ \tag{3.13}$$

for the subset $\mathcal{Z}^{(k)}$ in the k_{th} section. It can be seen that in the k_{th} section, the straight constraints of S_k are completely eliminated due to the choice of the range discretization (3.13). Then a finite number of admissible beam configurations in S_k that satisfy the discretized constraints are constructed. Combining the range discretization in all the sections, we can construct the finite number of admissible beam configurations for the discretized problem (3.9).

Now that the geometric constraints, domain, and range have been discretized, the problem expressed by Eq. (3.7) can be restated into the following discretized energy minimization form:
Find $\hat{\boldsymbol{\theta}}$ such that

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= arg \text{ min } \mathfrak{f}\left(\boldsymbol{\theta}, T_x, T_y\right) \\ &= arg \text{ min} \sum_{i=1}^{N-1} \left[\frac{EI_b}{2}\left(\frac{\theta_i - \theta_{i-1}}{\Delta s}\right)^2 - T_y \sin\theta_i + T_x\left[1 - \cos\theta_i\right]\right] \Delta s\end{aligned} \tag{3.14}$$

where

$$\theta_i = \theta_i^{(k_i)} + \beta_{k_i}, \;\; \sin\theta_i^{(k_i)} = \frac{y_{i+1}^{(k_{i+1})} - y_i^{(k_i)}}{\Delta s} \tag{3.15}$$

subject to (k_i is the section location of the i_{th} node)

$$-R_{limit} \leq y_i^{(k_i)}(s) \leq R_{limit}, \; i = 1, 2, \ldots, N, \; k_i \in (1, 2, \ldots, S) \tag{3.16}$$

The relationship between the global version stated in Equations (3.9-3.11) and the local version in Equations (3.14-3.16) can be seen by the fact that

$$\begin{cases} X_i = X_o^{k_i} + x_i^{(k_i)} \cos\beta_{k_i} - y_i^{(k_i)} \sin\beta_{k_i} \\ Y_i = Y_o^{k_i} + x_i^{(k_i)} \sin\beta_{k_i} + y_i^{(k_i)} \cos\beta_{k_i} \end{cases} \tag{3.17}$$

One can notice in Figure 3.16 that for the i_{th} link, ãĂŰ$Node$ãĂŮ$_i$ (start point) is located in S_k, but ãĂŰ$Node$ãĂŮ$_{i+1}$ (end point) belongs to the section S_{k+1}. In order to calculate the angle θ_i, the position of $Node_i$ (which was first labelled with respect to section S_k) should be labelled with respect to the section S_{k+1}. In this paper, the conversion is carried by the following equation.

$$\begin{cases} x_i^{(k_i+1)} = +\left(x_i^{(k_i)} - L_s^{k_i}\right)\cos\Delta\beta + y_i^{(k_i)} \sin\Delta\beta \\ y_i^{(k_i+1)} = -\left(x_i^{(k_i)} - L_s^{k_i}\right)\sin\Delta\beta + y_i^{(k_i)} \cos\Delta\beta \end{cases} \tag{3.18}$$

where $\Delta\beta = \beta_{k_i+1} - \beta_{k_i}$. Then θ_i can be calculated through the equation

$$\theta_i = \theta_i^{(k_i+1)} + \beta_{k+1}, \text{ where } \theta_i^{(k_i+1)} = \sin^{-1}\left(\frac{y_{i+1}^{(k_i+1)} - y_i^{(k_i+1)}}{\Delta s}\right) \tag{3.19}$$

3.2.4 Principle of discretization

As described in detail in [18], the Viterbi algorithm searches for the configuration with the minimum potential energy from the admissible beam configurations we constructed and finding a global minimum of the discretized problem is guaranteed (3.14). Thus, the accuracy of the result by Viterbi algorithm is mainly determined by the construction of the set of the admissible beam configurations. Therefore the discretization strategy directly decides the accuracy of the application of Viterbi algorithm to solve the beam deformation problem.

A proper discretization structure can be constructed by referring the following ideas:

1. A heuristic approach for the discretization strategy is that when carrying out the domain and range discretization, for most of buckling and post buckling problem, there are few cases that the beam is buckled backward; therefore it's better to construct a subset $\mathcal{Z}^{(k)} = \left\{y_1^{(k)}, y_2^{(k)}, \ldots, y_M^{(k)}\right\}$ for section S_k, which makes the angles of all possible configuration of segments equally distributed among a certain set, and in this set $\theta_i^{(k_i)} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$, where $\max\left(\theta_i^{(k_i)}\right) = \frac{\pi}{2}$ and $\min\left(\theta_i^{(k_i)}\right) = -\frac{\pi}{2}$.

2. The length of the segment should neither be too large nor too small. For most buckling problem of Euler beam, the length of each segment

$$\Delta s \approx \frac{2R_{limit}}{3} \tag{3.20}$$

   is a good choice for the first attempt to get the solution.

3. If there exist many large angle changes between two connective links in the beam configuration by Viterbi algorithm, we can decrease the length of segment and increase the number of elements of subset $\mathcal{Z}^{(k)}$ under the above two steps. However, we should notice that the discretization should not be too much fine especially for Range discretization; otherwise it will dramatically increase the cost of computation [18].

3.3 Markov structure

The discretized form of cantilever beam problem with constraints has a special discrete Markov structure that can be exploited by means of the well known Viterbi algorithm to find the minimum, which have been detailed in the work by Narayanan et al. [18, 19]. The Markov structure for this problem (3.14) is the same as that in the work by Narayanan et al. [18, 19]; therefore we will provide only a brief discussion about it.

Let S_i^{mn} be the state of the y displacements of the $Node_i$ and the $Node_{i+1}$ with respect to the local coordinate system. Then, the discretized problem (3.14) can be restated as:
Find $\hat{\boldsymbol{\theta}}$ such that

$$\hat{\boldsymbol{\theta}} = arg\,min \sum_{i=1}^{N-1} \left\{ \frac{EI_b}{2} \left[f\left(S_i^{mn}\right) - f\left(S_{i-1}^{lm}\right) \right] - T_y g_1\left(S_i^{mn}\right) + T_x g_2\left(S_i^{mn}\right) \right\} \Delta s \tag{3.21}$$

This Markov structure (dependence of g on S_{i-1}, S_i) can be exploited in the Viterbi algorithm. And in the following part, we will show the simulation results searched by Viterbi algorithm.

3.4 Results

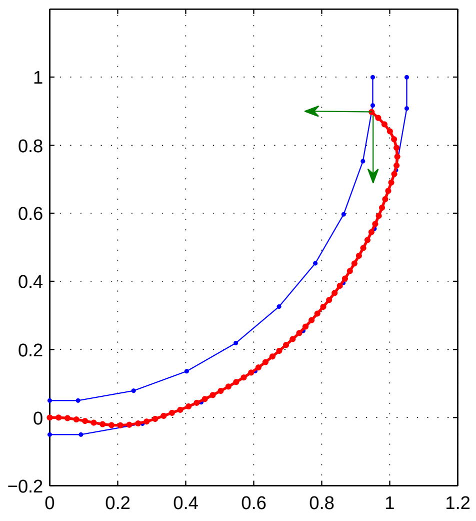
Here we present various results of the beam buckling problem with the constraints of a quarter circular channel by Viterbi algorithm. The geometries of beam and the curved constraints were set at $R = 1.0$, $R_{limit} = 0.05$, $r_o = 1.05$, $r_i = 0.95$, $L = \frac{\pi}{2}R$. The channel was discretized into $S = 10$ sections; for domain discretization the beam was divided into $N = 60$ equal links; in each section, the range was discretized according to equation (3.13) where M is taken as 300. Figure 3.17 shows the results obtained for 4 different load cases. Note that the end loads and the potential energy have been non-dimensionalized according to

$$T^\star = \left(T_x^\star, T_y^\star\right) = \left(\frac{T_x L^2}{EI_b}, \frac{T_y L^2}{EI_b}\right), \quad U = \frac{FL}{EI_b} \tag{3.22}$$

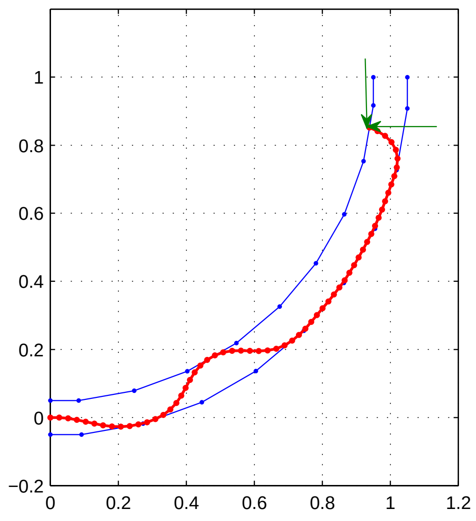
Figures 3.17a - 3.17c represent configurations subject to the compressive end load. The configurations agree very well with those observed with the qualitative experiment 3.1. Increasing the loads (in the negative direction) moves the right end of beam to the left and bottom, and the beam was buckled into the different patterns which agree with the patterns in experiment as shown in Figure 3.1. Figure 3.17d shows that the Viterbi algorithm can also work in the problem of pulling beam outside of the channel.

We note that the boundary conditions (BCs) at the right end of the beam in the qualitative experiment and in the simulation are different. In the experiment part, the BCs at right end are displacement with pinned end; while in simulation, the BCs at right end are forces. However, we should notice that any displacement boundary conditions in this paper can be consider as corresponding force (no moment) by applying forces until the appropriate displacement is obtained. So the results of the qualitative experiment can be used to qualitatively verify the validation of the technique based on Viterbi algorithm. The proof is shown in the following part.

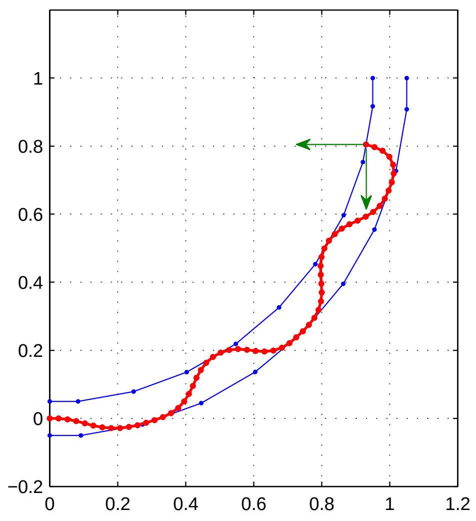
Let δR be the distance from the right end of the beam to the centreline of the channel; let L_d be the displacement of the right end of the beam moving along the centreline of channel (we define when $L_d = 0$, the beam lies along the centreline of the channel). In this way, the load for the beam in the qualitative experiment is the displacement L_d with additional boundary condition $\delta R = 0$; while the BC in the simulation is the end force $\mathbf{T}$. However, we should notice that in the qualitative experiment, only reaction force $\mathbf{T}$ can be created for the pinned boundary condition. Then we assume that a force $\mathbf{T}^p = (T_x^p, T_y^p)$ was carefully chosen, which makes the beam buckled exactly the same as that in the experiment; we name the beam buckling problem with end force $\mathbf{T}^p$ as equivalent buckling problem. Thus, the governing equation for the cantilever beam buckling problem in the equivalent buckling problem



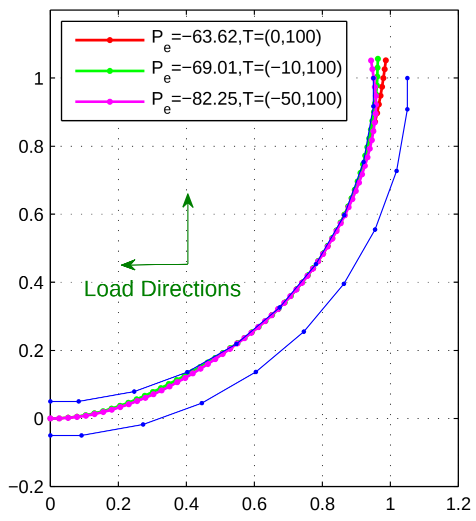
(a) $\mathbf{T} = (-350.0, -350.0)$, $PE = 67.201$



(b) $\mathbf{T} = (-500.0, -500.0)$, $PE = 93.760$



(c) $\mathbf{T} = (-700.0, -700.0)$, $PE = 119.76$



(d) $\mathbf{T} = (0.0; -10.0; -50.0; 100.0)$

Figure 3.17: The configurations of bucked beam calculated by Viterbi algorithm, for the case $R = 1$, $R_{limit} = 0.05$, $L = \frac{\pi}{2}R$, $S = 10$, $N = 60$, $M = 300$. The arrow represents the load direction. Figures (a), (b), (c) are for compressive loads (compare with experiment in Figure 3.1). Figure (d) is for tensile load (notice the contact with inner wall).

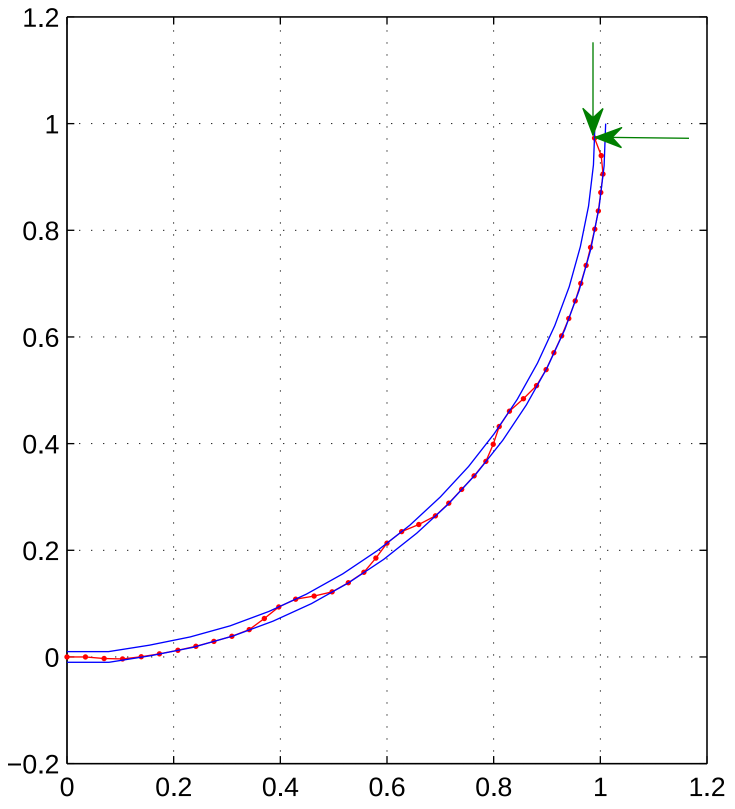


Figure 3.18: The configurations of bucked beam calculated by Viterbi algorithm, for the case $R = 1$, $R_{limit} = 0.01$, $L = \frac{\pi}{2}R$, $S = 20$, $N = 45$, $M = 100$. The arrow represents the load $\mathbf{T} = (-2500.0, -2500.0)$. The potential energy is $PE = 621.56$; the violation of the constraints is $Constrain_{max} = 3.327E - 4$.

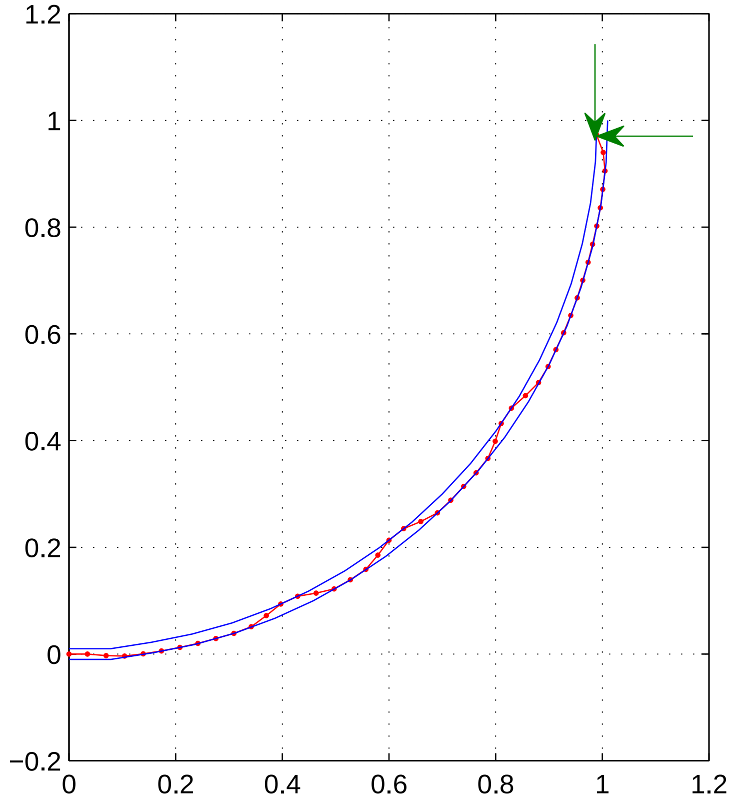


Figure 3.19: The configurations of bucked beam calculated by Active Set method with the initial condition obtained by Viterbi algorithm, for the case $R = 1$, $R_{limit} = 0.01$, $L = \frac{\pi}{2}R$, $N = 45$. The arrow represents the load $\mathbf{T} = (-2500.0, -2500.0)$. The potential energy is $PE = 636.27$; the violation of the constraints is $Constrain_{max} = 1.095E - 20$.

and simulation part should be the same in nature [13]: $EI_b \frac{d^2}{ds^2} Y(s) - T_y Y(s) + T_x X(s) = 0$, where the differences only lie in the material properties of the beam, the value of the end force at the right end of the beam and the parameters representative of the channel geometry. It means that both the problem in experiment and the problem in simulation have the same physical nature. So, we can conclude that the results of the experiment can be used to qualitatively verify the results by Viterbi algorithm in the simulations part.

Comparison with conventional minimization techniques The properties of Viterbi algorithm for beam deformation problem (independent of initial guesses) make it complementary to traditional algorithms (i.e. gradient descent method which is very dependent on initial guesses). According to the work by Doraiswamy et al. [18], conventional minimization techniques, such as active set method for constrained optimization [63], can further optimize the solution with initial condition got by Viterbi algorithm. In this paper, the solutions of the buckling problem (3.7) was also compared with an approach based on the active set method as implemented in Matlab (fmincon). Figure 3.18 shows the configuration of beam constrained in a narrow channel of width 0.02 under end load $\mathbf{T} = (-2500.0, -2500.0)$ by Viterbi algorithm. Figure 3.19 shows the result of the beam configuration got by active-set method under the same end load with the initial guess-the configuration got by Viterbi algorithm. The maximum of violation of constraints of Viterbi algorithm is $Constrain_{max} = 3.327E - 4$; while for active-set method with Viterbi initial guess, The maximum of violation of constraints is $Constrain_{max} = 1.095E - 20$. It is because the discretization error was introduced by discretization of range and constraints, and the active-set method further optimized the solution. However, this optimization of active-set method only works for the narrow channel; for the wide channel with width of $\Delta R = 0.1$, the active-set method fails to converge to the solution with the initial guess of the result by Viterbi algorithm, not to mention the random initial guess.

3.5 Convergence of the results to the continuous case

The method presented here, is based on a global search for the actual minimum of the energy, hence issues of whether or not it has converged to a global minimum do not arise. In other words, the Viterbi algorithm, by construction, is guaranteed to find a global minimum of the discretized problem (3.14). The only question of convergence that remains is whether will the discretized problem (3.14) converge to problem (3.9) and whether problem (3.9) converges to problem (3.7). Before we address these convergences, the following key was pointed out to help in identifying the different problem statements compared in this section.

- Equation (3.7) is the original statement of the problem with no discretization.
- Equation (3.9) is the statement of the problem with discretized domain.
- Equation (3.14) is the statement of the problem with discretized domain, range, and constraints.

It needs to emphasized that problem (3.14) was obtained from (3.7) by carrying out three kinds of discretization: discretization of domain, discretization of range, and discretization of constraints. Thus all these convergences should be established.

As shown in Figure 3.16, let the areas of the union, intersection of discretized channel constraints and the continuous channel constrains be $\mathbb{C}$, $\mathbb{D}$ respectively. Then the discretization error introduced by the discretizing constraints are the areas of $\mathbb{E} = \mathbb{C} - \mathbb{D}$, such as the area A_{abc} (eliminated from continuous constraints) and A_{def} (added to the continuous constraints). As the number of discretized sections S

increases, the areas $\mathbb{E}$ will decrease; specially, when S goes to infinite, the areas of $\mathbb{E}$ goes to zero, $\lim\limits_{S\to\infty} \mathbb{E} = 0$. For the range discretization, it is exactly the same as that in the work [18]. Assume that the number of elements in the set $\mathcal{Z}^{(k)}$ $(k = 1, \ldots, S)$ increase, the range is discretized finer; if $\mathcal{Z}^{(k)}$ is selected as equation3.13, in the limit ($\lim M \to \infty$), it can take any value between $-R_{limit}$ to $+R_{limit}$, and the range discretization can be taken as continuous. Thus, if both the number of sections S and the number of elements in the set $\mathcal{Z}^{(k)}$ go to infinite ($\lim S \to \infty$ and $\lim M \to \infty$), problem (3.14) will be identical to problem (3.9). Hence, convergence of (3.14) to (3.9) is established. In a similar manner we note that as $\Delta s \to 0$ and $N \to \infty$, the term $\dfrac{\theta_{i+1} - \theta_i}{\Delta s}$ tends to $\dfrac{d\theta}{ds}$ by definition of derivatives, and hence (3.9) converges to (3.7). These two step convergence process guarantee that the solution of the Viterbi algorithm converges to the solution of (3.7) as the constraints, range and domain are refined. A detailed convergence study has been carried out by Narayanan et al.[18][19] and therefore will not be pursued in detail here.

3.6 Simulation of static buckling of Euler beam subject to non-circular channel constraints

In this part, we will expand the application of Viterbi algorithm to the beam buckling problem with general channel constraints through an illustration of solving the problem shown in figure 3.20. The procedure to solve this problem is almost the same as that with circular channel constraints, and can be summarized as,

1. Design discretization strategies: (a) Discretizing the constraints. (b) Discretizing domain locally. (c) Discretizing range locally.

2. Construct the discretized problem in the form of Markov structure.

3. Employee the Viterbi algorithm to search the configuration of beam with lowest

energy.

There are only two differences in details for the above procedure. The first difference exist in the discretization of constraints, which are shown in figure 3.16 and figure 3.21. We should notice that the details of constraints discretization are pure geometric problems, and will not be discussed here. The second difference is the range discretization strategy- the finite subset $\mathcal{Z}^{(k)} = \left\{y_1^{(k)}, y_2^{(k)}, \ldots, y_M^{(k)}\right\}$ for the possible y-displacement of nodes in the k_{th} section will not be expressed by the equation (3.13) for circular channel, but expressed by the following equation,

$$y_m^{(k)} = \frac{W_k}{M}, m = 1, 2, \ldots, M \tag{3.23}$$

where W_k is the width of k_{th} section.

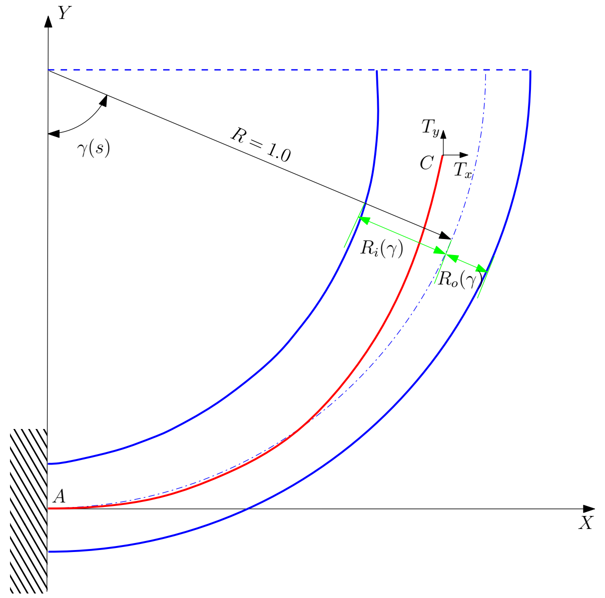


Figure 3.20: The figure shows a cantilever beam AC clamped at A, constrained to deform within a curve channel of radius $r_i = R - R_i(\gamma)$ and $r_o = R + R_o(\gamma)$ respectively. The curve line AC represents the current configuration of the beam. Loads T_x and T_y act at the free end C as shown.

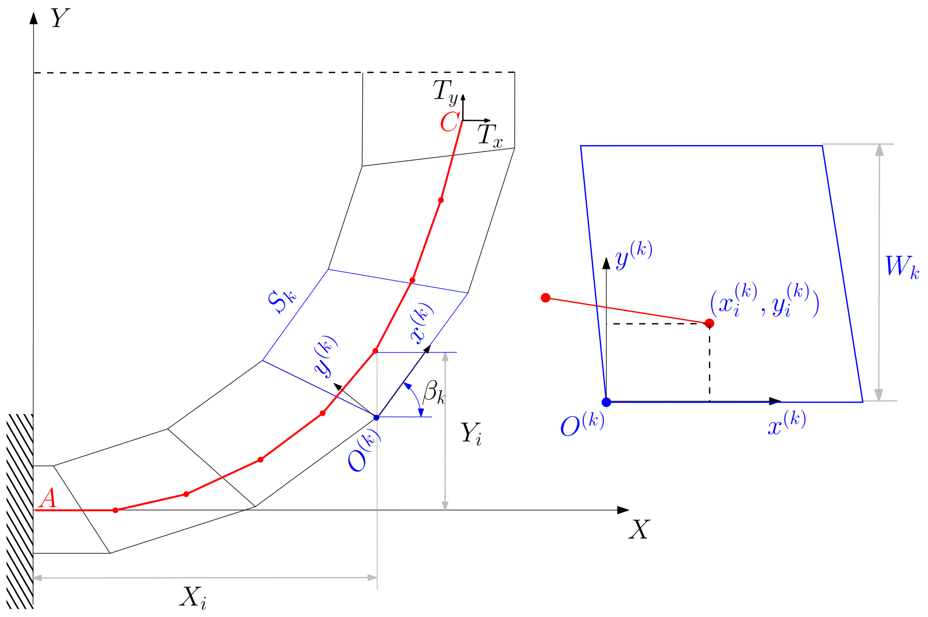


Figure 3.21: The discretized beam (the red dots in the lines AC) is constrained between several continuous straight parallel walls (the discrete curved channel constraints), and subjected to a load $\mathbf{T} = (T_x, T_y)$ at its free end C. The blue trapezoid represents the k_{th} section, and a local coordinate systems is assigned to each section.

The geometrics of the problem are set as $R = 1.0$, $R_i(r) = 0.05 + 0.025\pi\gamma$, $L = \dfrac{\pi}{2}R$ for the problem with divergent channel constraints (figure 3.22a); $R = 1.0$, $R_i(r) = 0.1 - 0.025\pi\gamma$, $L = \dfrac{\pi}{2}R$ for the problem with convergent channel constraints (figure 3.22b). The discretization strategies for both the problems are the same: the channels are discretized into $S = 8$ sections; for domain discretization the beam was divided into $N = 50$ equal links; in each section, the range was discretized according to equation (3.23) where $M = 40$. The results are shown in figure 3.22.

The simulation results are shown in figure 3.22 for the convergent and the divergent channel constraints. We can see that the buckling configurations as well as the potential energy of the beam differ from those obtained in a circular channel(figure 3.17b), with same load $\mathbf{T} = (-500, -500)$ acting at the right end of the beam.

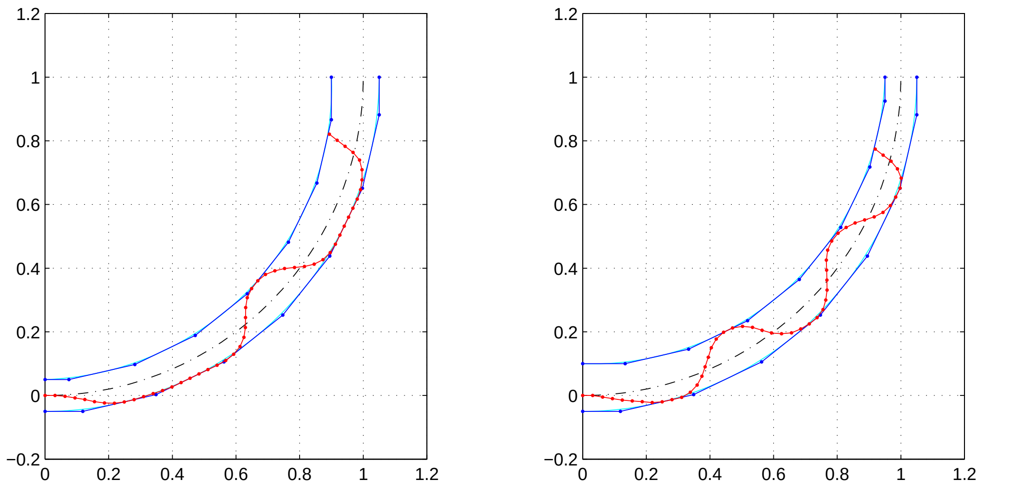

(a) $\mathbf{T} = (-500.0, -500.0)$, $PE = 77.2783$ (b) $\mathbf{T} = (-500.0, -500.0)$, $PE = 94.4431$

Figure 3.22: The configurations of bucked beam with the constraints of (a) divergent (b) convergent curve channels.

3.7 Conclusions

In this chapter, a direct search algorithm based on the Viterbi algorithm was presented to solve buckling of elastic beams with constraints of a curved channel by gainfully exploiting the Markov structure of the problem. The technique searched for the configuration with minimum potential energy among a set of admissible configurations for the beam. Accuracy of the Viterbi algorithm applied to the solution of beam bucking depends on the way that domain, range and constraints are discretized. The method presented here can easily be extended to any curved channels, as long as the constraints are discretized into pieces of straight channel constraints and the ranges for each discretized section are properly discretized.

Since the method base on Viterbi algorithm searches all the discrete admissible configurations we constructed and store all the possible configurations with minimum potential energy at each stage, both the computation cost (for searching) and memory (for storing all possible configurations) are very high. Particularly, an increase on dimension can cause exponential growth on the computation cost and memory requirement. We are still exploring the limits of this novel (to us) approach. For the 3-D case presently we are facing the "curse of dimensionality" and our simple implementation seems to be too slow. Moreover, the incapability of handling with torsion behavior is a factor that limits the application of Viterbi algorithm to simulate rod like object.

4. SIMULATION OF DYNAMIC RESPONSE OF KIRCHHOFF RODS AT REAL-TIME RATE

In this chapter, we report on results obtained with the model based on a discretized elastic rod [49] based on Kirchhoff theory of elastic rods, which overcome many of the limitations of the current physics based approaches especially with regard to long term stability, inextensibility, collision detection and response, and internal dissipation, in the dynamic simulation of thread. The uniqueness of our approach stems from the fact that we

i) Overcome the long term stability issues by adapting the discrete variational integrator technique to develop an explicit time integration scheme that is fast but is stable over over long times.

ii) Develop a new technique based on Picard iterations to enforce inextensibility exactly at every time step without sacrificing speed and without the need for penalty function (i.e. stiff extension springs)

iii) Introduce a new technique for dealing with string entanglement to prevent inter-penetration and pass through that does not suffer from "bounceback" or "jitter" by using continuous penalty force. This is a difficult problem for one dimensional models (with zero thickness) since , within a single time step a portion of the rod can pass thorough a different portion and collisions or inter-penetration may not be detected at all.

iv) Develop a new way of incorporating internal string inelasticity as well as external air damping, which is critical for simulating realistic motions with rapid cessation of motion

A key element of the work is that rather than working with directors that are constrained in lengths and relative angles, or in terms of Seret-Frenet Frames, we work with the Bishop or untwisted frame, following the work by Bergou et al. [49]. Also we eliminate the torsional dynamics since it is much faster than the dynamics due to bending, so that *the resulting equations are entirely in terms of the positions of the nodal points only* this is a huge advantage in speeding up the computations since we are not dealing with 15 equations for the 15 unknowns [64] (as is conventional in Cosserat rod theories). We illustrate these capabilities by considering the dynamics of the formation of plectonemes wherein the bend-twist coupling as well as self interaction are critical.

4.1 Mathematical model

A thread can be modeled as a circular rod with a diameter much smaller than its length and is termed a one-dimensional object. When subjected by input forces and/or moments, the rod takes on three-dimensional configurations. To describe the configurations of the rod in space and time requires the knowledge of the position of the rod centerline and the orientation of the cross section. The rod may bend, twist or assume a distinct coiling effect but it cannot stretch (constraint of inextensibility).

4.1.1 Continuous model

Since this work is inspired by the work of Bergou et al.[49], our notation mainly follow their work.

Framed-curve representation The geometric dimension of the cross section of the Kirchhoff rod with mass M is very small compared with its length; thus we assume it is an one dimensional elastic object, which configuration is described by its centerline. Following the Kirchhoff theory of elastic rods, the configuration

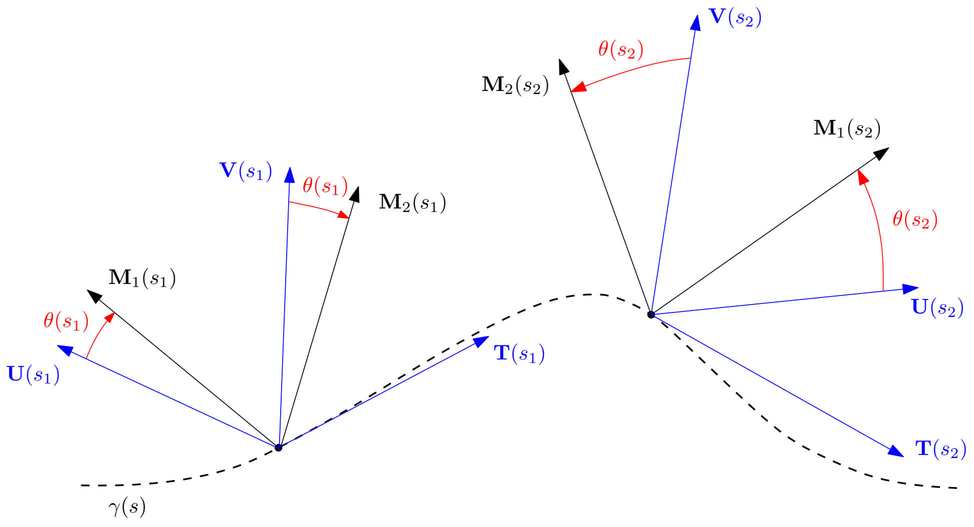


Figure 4.1: The configuration of an Kirchhoff rod is represented by a curve $\boldsymbol{\gamma}(s)$ and a material frame $\{\mathbf{T}(s), \mathbf{M_1}(s), \mathbf{M_2}(s)\}$. The twist can be measured through the angle difference $\theta(s)$ between the material frame and the Bishop frame $\{\mathbf{T}(s), \mathbf{U}(s), \mathbf{V}(s)\}$.

of the rod is represented by an adapted framed curve $\Gamma = \{\boldsymbol{\gamma}; \mathbf{T}, \mathbf{M_1}, \mathbf{M_2}\}$ over a parameter $s \in [0, L]$, where L is the length of the rod (see Fig. 4.1). Here $\boldsymbol{\gamma}(s)$ is an arc length parameterized curve in $\mathbb{R}^3$ describing the rod's *centerline*; a right-handed orthonormal coordinate system $\{\mathbf{T}(s), \mathbf{M_1}(s), \mathbf{M_2}(s)\}$ is assigned to each point on the centerline and named as *material frame*. The material frame satisfies $\mathbf{T}(s) = \frac{d}{ds}\boldsymbol{\gamma}(s)$, i.e., it is adapted to the centerline such that the first material axis is tangent to the curve; $\mathbf{T}$ contains the bending information and $\mathbf{M_1}$ (or $\mathbf{M_2}$) contains the twisting information. In the following part, more details about the material frame will be given.

Darboux Vector Because the orthonormal material frame $\{\mathbf{T}(s), \mathbf{M_1}(s), \mathbf{M_2}(s)\}$ is normalized, the derivative of its components (i.e. $\mathbf{T}$) with respect to arc length $s \in [0, L]$ must be orthogonal to the component itself (i.e. $\mathbf{T}$). Thus the derivative can be represented by the other components (i.e. $\mathbf{M_1}$ and $\mathbf{M_2}$),

$$\frac{\partial \mathbf{T}(s)}{\partial s} = a\mathbf{M_1}(s) + b\mathbf{M_2}(s) \tag{4.1}$$
$$\frac{\partial \mathbf{M_1}(s)}{\partial s} = c\mathbf{T}(s) + d\mathbf{M_2}(s) \tag{4.2}$$
$$\frac{\partial \mathbf{M_2}(s)}{\partial s} = e\mathbf{T}(s) + f\mathbf{M_1}(s) \tag{4.3}$$

with appropriate values for a, b, c, d, e and f. Since

$$\begin{array}{rcr} \frac{\partial}{\partial s}\left(\mathbf{T}(s)\cdot\mathbf{M_1}(s)\right) = 0 & & \frac{\partial \mathbf{T}(s)}{\partial s}\mathbf{M_1}(s) = -\frac{\partial \mathbf{M_1}(s)}{\partial s}\mathbf{T}(s) \\ \frac{\partial}{\partial s}\left(\mathbf{M_1}(s)\cdot\mathbf{M_2}(s)\right) = 0 & \Rightarrow & \frac{\partial \mathbf{M_1}(s)}{\partial s}\mathbf{M_2}(s) = -\frac{\partial \mathbf{M_2}(s)}{\partial s}\mathbf{M_1}(s) \\ \frac{\partial}{\partial s}\left(\mathbf{M_2}(s)\cdot\mathbf{T}(s)\right) = 0 & & \frac{\partial \mathbf{M_2}(s)}{\partial s}\mathbf{T}(s) = -\frac{\partial \mathbf{T}(s)}{\partial s}\mathbf{M_2}(s) \end{array}$$

it is quite straight to find that the constants a, b, c, d, e and f in equations (4.1)-(4.3) can be reduced to three variables, ω_1, ω_2 and m_t, such that

$$\frac{\partial \mathbf{T}(s)}{\partial s} = \omega_1\mathbf{M_1}(s) + \omega_2\mathbf{M_2}(s) \tag{4.4}$$
$$\frac{\partial \mathbf{M_1}(s)}{\partial s} = -\omega_1\mathbf{T}(s) + m_t\mathbf{M_2}(s) \tag{4.5}$$
$$\frac{\partial \mathbf{M_2}(s)}{\partial s} = -\omega_2\mathbf{T}(s) - m_t\mathbf{M_1}(s) \tag{4.6}$$

Therefore, we can define a *Darboux vector*, $\mathbf{\Omega}(s) := m_t\mathbf{T}(s) - \omega_2\mathbf{M_1}(s) + \omega_1\mathbf{M_2}(s)$, such that the equations (4.4)-(4.6) can be rewrite as,

$$\frac{\partial \mathbf{T}(s)}{\partial s} = \mathbf{\Omega}(s) \times \mathbf{T}(s) \tag{4.7}$$
$$\frac{\partial \mathbf{M_1}(s)}{\partial s} = \mathbf{\Omega}(s) \times \mathbf{M_1}(s) \tag{4.8}$$
$$\frac{\partial \mathbf{M_2}(s)}{\partial s} = \mathbf{\Omega}(s) \times \mathbf{M_2}(s) \tag{4.9}$$

Substituting $\mathbf{\Omega}(s)$ into the above equations, we can get

$$\omega_1 = \mathbf{T}' \cdot \mathbf{M_1},\ \omega_2 = \mathbf{T}' \cdot \mathbf{M_2} \quad \text{and} \quad m_t = \mathbf{M_1}' \cdot \mathbf{M_2} \tag{4.10}$$

Notice that the prime $y' = y(s,t)$ denotes spacial derivative $\frac{\partial y}{\partial s}$, while the dot $\dot{y}(s,t)$ denotes the temporal derivative $\frac{\partial y}{\partial t}$. Since $\mathbf{T}'$ is the centerline's *curvature* vector, $\boldsymbol{\omega} = [\omega_1, \omega_2] = [\mathbf{T}' \cdot \mathbf{M_1}, \mathbf{T}' \cdot \mathbf{M_2}]$ represents the rod's curvature vector expressed in material coordinates and measure the bending of material frame. Also, the equation (4.5) shows that $\mathbf{M_1}'$ exist in two directions, $\mathbf{T}$ and $\mathbf{M_2}$, so $m_t = \mathbf{M_1}' \cdot \mathbf{M_2}$ measures the rotation rate of $\mathbf{M_1}$ around the tangent $\mathbf{T}$ with respect to the arc length s and thus measure the twist of material frame [49].

We can now obtain a more explicit conclusion, the space propagation of the material frame along the centerline of the rod can be described in terms of the Darboux vector, $\mathbf{\Omega} := m_t\mathbf{T} - \omega_2\mathbf{M_1} + \omega_1\mathbf{M_2}$, by the equations $\mathbf{T}' = \mathbf{\Omega} \times \mathbf{T}$, $\mathbf{M_1}' = \mathbf{\Omega} \times \mathbf{M_1}$, $\mathbf{M_2}' = \mathbf{\Omega} \times \mathbf{M_2}$, where m_t measures the twist, ω_1 and ω_2 measure the bending.

Elastic energy The Kirchhoff theory of elastic rods assigns an elastic energy, $E(\Gamma)$ to any adapted frame curve Γ. Since we assume that the rod is inextensible, we do not consider the stretching energy and the total elastic energy are contributed by bending and twisting energy,

$$E(\Gamma) = E_{bend}(\Gamma) + E_{twist}(\Gamma) \tag{4.11}$$

The inextensibility of the rod is achieved by Lagrangian multipliers. Of course, it is straightforward to drop this assumption by also including a stretching term to simulate extensive elastic rod.

In this dissertation, we assume that the rod is naturally straight and isotropic. Therefore, the ***bending energy*** takes the simple form [49],

$$E_{bend}(\Gamma) = \frac{1}{2}\int_0^L k_b \boldsymbol{\omega}^T \boldsymbol{\omega}\ \mathrm{d}s \ = \ \frac{1}{2}\int_0^L k_b \kappa^2\ \mathrm{d}s \tag{4.12}$$

where the $\boldsymbol{\omega} = (\omega_1, \omega_2)^T$ represents the centerline curvature vector expressed in the material frame coordinates; $\kappa = |\boldsymbol{\omega}|$ is the curvature of the centerline; and k_b is the bending stiffness of the rod.

Since m_t denote the twist of the material frame about the centerline, the ***twisting energy*** is given by [49],

$$E_{twist}(\Gamma) = \frac{1}{2}\int_0^L k_t m_t(s)^2\ \mathrm{d}s \tag{4.13}$$

where k_t is the rod's torsional stiffness. The formula $m_t = \mathbf{M_1}' \cdot \mathbf{M_2}$ gives an expression for the twist in terms of material vectors immersed in ambient space. According to the discrete elastic model based on the theory of Kirchhoff rod[49], the torsion is measured through the angle difference between material frame and Bishop frame, we now seek this equivalent expression measuring twist.

Bishop frame The *Bishop frame* $\{\mathbf{T}(s), \mathbf{U}(s), \mathbf{V}(s)\}$ for the given centerline is an adapted frame with zero twist uniformly, *i.e.*, $m_t(s) = 0,\ s \in [0, L]$. Therefore the Bishop frame for a given rod is equivalent to a material frame of a curve, which has the same position of the given rod's centerline, with zero twist. The space propagation of the Bishop frame along the arc length of the centerline can be described in terms of its Darboux vector with twisting set to zero, $\boldsymbol{\Omega} = -\omega_2\mathbf{U} + \omega_1\mathbf{V}$, through the equations

$$\mathbf{T}' = \boldsymbol{\Omega} \times \mathbf{T}, \quad \mathbf{U}' = \boldsymbol{\Omega} \times \mathbf{U} \quad \text{and} \quad \mathbf{V}' = \boldsymbol{\Omega} \times \mathbf{V}.$$

So if a rod is not twisted with $m_t(s) = 0,\ s \in [0, L]$, then its material frame and Bishop frame are the same. We should notice that the assignment of an adapted frame to one point on the curve uniquely fixes the Bishop frame throughout the curve. Usually the Bishop frame is assigned at the start point ($s = 0$) of the curve.

Since $\mathbf{T}' = \boldsymbol{\Omega} \times \mathbf{T}$, we know

$$\mathbf{T} \times \mathbf{T}' = \mathbf{T} \times (\boldsymbol{\Omega} \times \mathbf{T}) = \boldsymbol{\Omega}\,(\mathbf{T} \cdot \mathbf{T}) - \mathbf{T}\,(\mathbf{T} \cdot \boldsymbol{\Omega}) = 1 \cdot \boldsymbol{\Omega} - 0 \cdot \mathbf{T} = \boldsymbol{\Omega}.$$

Recall that $\mathbf{T}'$ is the centerline's curvature vector; thus, we can get $\boldsymbol{\Omega} = \boldsymbol{\kappa}\mathbf{b}$, where $\boldsymbol{\kappa}\mathbf{b} = \mathbf{T} \times \mathbf{T}'$ is the *curvature binormal* along the centerline.

The Darboux vector of the Bishop frame serves to define *parallel transport*. We parallel transport a vector $\mathbf{x}$ from one point on the centerline to another by integrating the equation $\mathbf{x}' = \boldsymbol{\kappa}\mathbf{b} \times \mathbf{x}$. Thus, infinitesimally, parallel transport corresponds to a *rotation about the binormal*, an important concept that we will use in our discrete model. Parallel transport keeps the tangential component of $\mathbf{x}$ tangential, and evolves the cross-sectional component of $\mathbf{x}$ via a tangential velocity, in particular without rotating the cross-section about the centerline.

The Bishop frame allows for a simple parameterization of the material frame[51]. Let $\theta(s)$ be the scalar function that measures the rotation about the tangent of the material frame relative to the Bishop frame (see Fig. 4.1). The following relation between Bishop frame and material frame must exist,

$$\mathbf{M_1}(s) = \mathbf{R}(\theta(s))\mathbf{U}(s)$$

where $\mathbf{R}(\theta(s)) \in \mathbb{R}^3 \times \mathbb{R}^3$ is the rotation around $\mathbf{T}(s)$ with angle $\theta(s)$. This implies

$$\frac{\partial \mathbf{M_1}(s)}{\partial s} = \left(\frac{\partial}{\partial s}\mathbf{R}(\theta(s))\right)\mathbf{U}(s) + \mathbf{R}(\theta(s))\frac{\partial \mathbf{U}(s)}{\partial s} \tag{4.14}$$

Since the Darboux vector for the Bishop frame implies $m_t = 0$, recalling the equation (4.10), we can get

$$\frac{\partial \mathbf{U}(s)}{\partial s} \cdot \mathbf{V} = 0$$

Therefore the second summand of equation (4.14) is aligned to $\mathbf{T}(s)$ and does not contribute to the twist m_t for material frame. The first summand can be reformed as,

$$\begin{aligned}\left(\frac{\partial}{\partial s}\mathbf{R}(\theta(s))\right)\mathbf{U}(s) &= \frac{\partial \theta(s)}{\partial s}\frac{\partial \mathbf{R}}{\partial \theta}\mathbf{U}(s) \\ &= \frac{\partial \theta(s)}{\partial s}\mathbf{R}(\theta(s))\mathbf{R}(\pi/2)\mathbf{U}(s) \\ &= \frac{\partial \theta(s)}{\partial s}\mathbf{R}(\theta(s))\mathbf{V}(s) = \frac{\partial \theta(s)}{\partial s}\mathbf{M_2}\end{aligned} \tag{4.15}$$

Substituting equations (4.14) and (4.15) into the equation (4.10), we can get,

$$m_t = \frac{\partial \theta(s)}{\partial s} \tag{4.16}$$

Hence, we can write the twisting energy as

$$E_{twist}(\Gamma) = \frac{1}{2}\int_0^L k_t \left(\frac{\partial \theta(s)}{\partial s}\right)^2 \, \mathrm{d}s \tag{4.17}$$

Observe that we have expressed the elastic energy of Kirchhoff rods by two dominant players: the position of the centerline, $\boldsymbol{\gamma}(s)$, and the angle of rotation, $\theta(s)$, between the Bishop and the material frame.

4.1.2 Discretized model

Discrete framed curves The centerline of the rod is discretized along $\boldsymbol{\gamma}(s)$ into $N_L \in \mathbb{N}$ segments, and the masses (m_i) are concentrated on each vertices. Therefore the discrete framed curve, Γ, consists of a centerline comprised of (N_L+1) vertices $\mathbf{x} = (\mathbf{x}_1 \ldots \mathbf{x}_{N+1})$ and N_L straight edges $\mathbf{e}_1 \ldots \mathbf{e}_N$ such that $\mathbf{e}_i = \mathbf{x}_{i+1} - \mathbf{x}_i$, together with an assignment of material frames $\mathbf{M}^i = \{\mathbf{T}^i, \mathbf{M}_1^i, \mathbf{M}_2^i\}$ and Bishop frames $\mathbf{B}^i = \{\mathbf{T}^i, \mathbf{U}^i, \mathbf{V}^i\}$ per edge (see Fig. 4.2), where $\mathbf{T}^i = \mathbf{e}_i/|\mathbf{e}_i|$ is the unit tangent vector per edge. We naturally assign frames to edges, rather than to vertices. The inextensibility constraints of the discrete surgical rod then can be expressed by

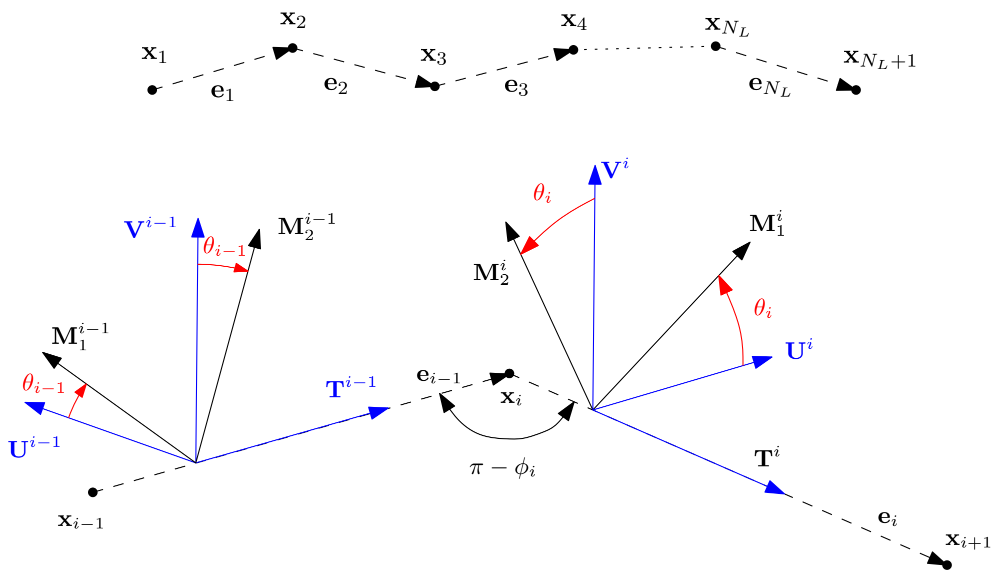


Figure 4.2: Discrete framed curves.

the following equation,

$$\mathbf{\Psi}(\mathbf{x}) = \begin{bmatrix} (\mathbf{x}_2 - \mathbf{x}_1)^2 - l_1^2 \\ (\mathbf{x}_3 - \mathbf{x}_2)^2 - l_2^2 \\ \vdots \\ (\mathbf{x}_{(N_L+1)} - \mathbf{x}_{N_L})^2 - l_{N_L}^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{4.18}$$

where $l_i = |\mathbf{x}_{i+1} - \mathbf{x}_i|$ is the length of the i_{th} discrete straight segment, $i = (1, 2, \ldots, N_L - 1, N_L)$.

Discrete bending energy When deriving the discrete bending energy, we adopt the same guiding principle as Bergou et al.[49], which is to seek a viewpoint that builds on the same *geometric principles* as the corresponding smooth theory, though we want to simulate the behavior as physical as possible. Since each edge is straight, it follows that discrete curvature is naturally associated with vertices. Letting ϕ_i denote the turning angle between two consecutive edges (see Fig. 4.2). We define (integrated) curvature by

$$\boldsymbol{\kappa}_i = 2\tan\frac{\phi_i}{2}$$

We now have all the pieces to assemble the bending energy of a discrete naturally straight, isotropic rod:

$$E_{bend} = \frac{1}{2}\sum_{i=1}^{N_L} k_b \left(\frac{\boldsymbol{\kappa}_i}{\bar{l}_i/2}\right)^2 \frac{\bar{l}_i}{2} = \sum_{i=1}^{N_L} \frac{k_b(\boldsymbol{\kappa}_i)^2}{\bar{l}_i}$$

where $\bar{l}_i = |\mathbf{e}_{i-1}| + |\mathbf{e}_i|$. Since

$$\begin{aligned}\left(\tan\frac{\phi_i}{2}\right)^2 &= \frac{\sin^2(\phi_i/2)}{\cos^2(\phi_i/2)} = \frac{1-\cos^2(\phi_i/2)}{\cos^2(\phi_i/2)} = \frac{1}{\cos^2(\phi_i/2)} - 1 \\ &= \frac{1}{\left(2\cos^2(\phi_i/2) - 1\right)/2 + 1/2} - 1 = \frac{2}{\cos\phi_i + 1} - 1\end{aligned}$$

The bending elastic energy can be further simplified as

$$E_{bend} = \sum_{i=1}^{N_L} \frac{k_b(\boldsymbol{\kappa}_i)^2}{\bar{l}_i} = \sum_{i=1}^{N_L} \frac{4k_b}{\bar{l}_i} \left(\frac{2}{1+\cos\phi_i} - 1 \right) \tag{4.19}$$

Notice that $\boldsymbol{\kappa}_i \longrightarrow \inf$ as incident edges bend toward each other ($\phi_i = \pi$), so this measure of curvature will penalize sharp kinks in the rod. In this sense, the discrete bending energy is properly defined.

Discrete twist energy Finding a discretized energy based on the continuous one (4.17) is quite straight. Here we directly adopt the discrete twist energy by Bergou et al.[49],

$$E_{twist} = \frac{1}{2} \sum_{i=1}^{N_L} k_t \frac{(\theta_i - \theta_{i-1})^2}{\bar{l}_i/2} = \sum_{i=1}^{N_L} \frac{k_t m_{t_i}^2}{\bar{l}_i} \tag{4.20}$$

where the scalar $m_{t_i} = \theta_i - \theta_{i-1}$ is the *discrete twist.* The variable m_{t_i} measures the angle between the result frame of parallel transporting the material frame from edge $\mathbf{e}_{i-1}$ to $\mathbf{e}_i$ and the material frame at edge $\mathbf{e}_i$ itself.

Discrete elastic energy Since we assume that the rod is a naturally straight and isotropic rod, the twist energy by equation (4.20) can be further simplified. In the work by Bergou et al.[49], they observe that the twist waves propagate much faster than bending waves and thus assume that twist waves propagate instantly. Therefore they get the conclusion that at any instant in time, the material frames are the minimizer of elastic energy, subject to any given boundary conditions on the material frame,

$$\frac{\partial E(\Gamma)}{\partial \theta_j} = 0 \tag{4.21}$$

The elastic energy of the rod include two parts, bending energy and twist energy, as shown in equation (4.11). While only twist energy depends on θ_j variables, according to the equations (4.11), (4.20), and (4.21), it is quite straight to get,

$$\frac{m_{t_i}}{\bar{l}_i} = \frac{\theta_{N_L} - \theta_1}{2\bar{L}} = constant \tag{4.22}$$

where $2\bar{L} = \sum_{i=1}^{N_L} \bar{l}_i$, the point-wise twist m_{t_i}/l_i depends only on the boundary conditions: the difference of angles of the end edges, $\theta_N - \theta_1$. Therefore, the elastic energy for our discrete rod model is

$$E(\Gamma) = \sum_{i=1}^{N_L} \frac{k_b}{\bar{l}_i} \left(\frac{1}{1 + \cos\phi_i} - \frac{1}{2} \right) + \frac{k_t(\theta_{N_L} - \theta_1)^2}{2\bar{L}} \tag{4.23}$$

4.1.3 Air drag

Because the density of the surgical thread is very small, one must take into account the drag caused by the motion of rod in air. Here (as in [65]), we assume that the air drag is an external force proportional to the rod velocity,

$$\mathbf{F}_a(\mathbf{v}_i) = -k_{air}(\mathbf{v_i}^T\mathbf{v_i})\hat{\mathbf{v}}_i \tag{4.24}$$

where k_{air} is the drag coefficient, and $\hat{\mathbf{v}}_i = \dfrac{\mathbf{v_i}}{|\mathbf{v_i}|}$.

4.1.4 Internal dissipation

Even though the structure is assumed elastic, the dissipation caused by the internal friction of the rod may not be neglected. In [46], Spillmann et al. calculate the relative angular velocity of each vertex to account for the dissipation, a method that requires a lengthy computation. In this paper, since we assume the rod is inextensible and the evolution of the twist of the rod is instantaneously, we neglect the dissipation caused by stretch and torsion, and only consider the dissipation due to bending,

$$P_d(\Gamma) = \sum_{i=2}^{N_L} \frac{1}{2} k_d \dot{\beta}_i^2 \tag{4.25}$$

where $\beta_i = \cos\phi_i$ measures the bending of the discrete rod (see Figure 4.2). Then the force caused by the bending dissipation is expressed as,

$$\mathbf{F_b} = \frac{\partial P_d(\Gamma)}{\partial \dot{\mathbf{x}}} \tag{4.26}$$

To find the expression of forces caused by internal dissipation $\mathbf{F_b}$, we first derive the individual summands,

$$\begin{aligned}
\mathbf{S}_i^W &:= k_d \dot{\beta}_{i-1} \frac{\partial \dot{\beta}_{i-1}}{\partial \dot{\mathbf{x}}_i} = k_d \dot{\beta}_{i-1} \frac{\partial \beta_{i-1}}{\partial \mathbf{x}_i} = k_d \dot{\beta}_{i-1} \frac{\mathbf{x}_{i-1} - \mathbf{x}_{i-2}}{l_{i-2} \cdot l_{i-1}} \\
\mathbf{S}_i^O &:= k_d \dot{\beta}_i \frac{\partial \dot{\beta}_i}{\partial \dot{\mathbf{x}}_i} = k_d \dot{\beta}_i \frac{\partial \beta_i}{\partial \mathbf{x}_i} = k_d \dot{\beta}_i \frac{\mathbf{x}_{i+2} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{l_{i-1} \cdot l_i} \\
\mathbf{S}_i^E &:= k_d \dot{\beta}_{i+1} \frac{\partial \dot{\beta}_{i+1}}{\partial \dot{\mathbf{x}}_i} = k_d \dot{\beta}_{i+1} \frac{\partial \beta_{i+1}}{\partial \mathbf{x}_i} = k_d \dot{\beta}_{i+1} \frac{\mathbf{x}_{i+2} - \mathbf{x}_{i+1}}{l_i \cdot l_{i+1}}
\end{aligned} \tag{4.27}$$

With the definition of above terms, the forces obtained through equation (4.26) can be as,

$$\mathbf{F}_b = \left[\mathbf{F}_1^b, \mathbf{F}_2^b, \ldots, \mathbf{F}_N^b\right]^T \tag{4.28}$$

where

$$\begin{aligned}
&\mathbf{F}_1^b = \mathbf{S}_1^E; \qquad \mathbf{F}_2^b = \mathbf{S}_2^O + \mathbf{S}_2^E \\
&\mathbf{F}_i^b = \mathbf{S}_i^W + \mathbf{S}_i^O + \mathbf{S}_i^E, \qquad i = 3 \ldots N-2 \\
&\mathbf{F}_{N-1}^b = \mathbf{S}_{N-1}^O + \mathbf{S}_{N-1}^W; \qquad \mathbf{F}_N^b = \mathbf{S}_N^W
\end{aligned}$$

4.2 Variational integrator

Since the material frame is always updated to be in quasi-static equilibrium (equation 4.21), we only need to update the centerline based on the forces derived above. The equations of motion can be constructed like this

$$\mathbf{M}\ddot{\mathbf{x}} = -\frac{\mathrm{d}E(\Gamma)}{\mathrm{d}\mathbf{x}}$$

, where $\mathbf{M}$ is a $3(N_L+1) \times 3(N_L+1)$ mass matrix associated to vertices of discrete rod. Usually, this equation is first discretized and then solved by explicit or implicit integrators. The simulations are expected to run for a relatively long period of time so it is essential that the algorithm is able to conserve energy. However, the explicit integrator require small time step; implicit integrator is robust but require complex computation of Jacobi matrix of forces [9]; most importantly, none of the explicit or implicit integrator can guarantee the conservation of energy and momentumy[66]. The discrete variational integrators derived from Lagrangian mechanics have the property of being symplectic and thus guarantee the conservation of energy and momentum [67, 66]. This indicates that discrete variational integrators can be well suited to develop robust algorithms for a system required long run stability [68]. So we employ variational integrator to simulate the rod and the details about variational integrator will be shown in the following part. A scheme to obtain the numerical solution for the deformations in space and time of a very thin and long circular rod (i.e. a thread) is presented in the following part.

4.2.1 Advantages of variational integrator

Our object of this dissertation is to develop a simulator that can be used to train surgeons' suture skills. A beginner may take several minutes or even more than ten minutes to tie a knot. Therefore to avoid unexpected behavior in the simulation, long time stability of the simulation is very important, which requires the energy and momenta conservation are considered. The variational integrator exactly preserve momenta and have excellent longtime energy stability. These properties make them ideal for simulating physical systems which are either conservative or near-conservative. In addition, the variational methodology allows one to easily and cleanly derive good integrators. Moreover, by employing variational integrator, it is very straightforward to derive the constrained discrete equations to achieve the inextensibility of the rod through Lagrangian multipliers, while still maintaining the conservation properties as the unconstrained discrete equations[66]. Hence, the variational integrator is an ideal tool to solving the discrete rod.

4.2.2 Continuous time Lagrangian mechanics

Any integrator which is the discrete Euler-Lagrange equation for some discrete Lagrangian is called a variational integrator[66]. To better explain the implications of discrete Euler-Lagrange equation, in the following we briefly review the Lagrangian formulation of the mechanics of a conservative system, and then we mimic this process at the discrete level to construct variational integrators.

Consider the Lagrangian system $\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}})$, where $\mathbf{q} = (q^1, q^2, \ldots, q^n)$ is a point in the configuration space Q. In Lagrangian mechanics the trajectories of the system should obey Hamilton's principle, namely, we seek paths $\mathbf{q}$ for which the action functional

$$S(\mathbf{q}(t)) = \int_{t_a}^{t_b} \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) dt$$

is stationary when compared with other paths with the same endpoints at times t_a

and t_b. This gives that,

$$\begin{aligned}\delta S(\mathbf{q}) = \delta \int_{t_a}^{t_b} \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) dt &= \int_{t_a}^{t_b} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}} \cdot \delta \mathbf{q} + \frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \cdot \delta \dot{\mathbf{q}} \right) dt \\ &= \int_{t_a}^{t_b} \left(\frac{\partial \mathcal{L}}{\partial \mathbf{q}} - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \right) \right) \cdot \delta \mathbf{q} \, dt + \left[\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}} \cdot \delta \mathbf{q} \right]_{t_a}^{t_b}\end{aligned}$$

where we have used integration by parts. The final term is zero because we assume that $\delta \mathbf{q}(a) = \delta \mathbf{q}(b) = 0$. Requiring that the variations of the action be zero ($\delta S(\mathbf{q}) = 0$) for all δq implies that the integrand must be zero for each time t, giving the well-known *Euler-Lagrange equations*,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{q}}(\mathbf{q}, \dot{\mathbf{q}}) - \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}}(\mathbf{q}, \dot{\mathbf{q}}) \right) = 0$$

4.2.3 Discrete time Lagrangian mechanics

We will now see how discrete variational mechanics (including the inextensibility constraints) performs an analogue of the above derivation. We now consider a discrete Lagrangian $\bar{\mathcal{L}}_d(\mathbf{x}^k, \dot{\mathbf{x}}^k)$, approximating the action integral during time period of $[k \cdot h, (k+1)h]$,

$$\bar{\mathcal{L}}_d(\mathbf{x}^k, \dot{\mathbf{x}}^k) = h \bar{\mathcal{L}}(\mathbf{x}^k, \dot{\mathbf{x}}^k) \approx \int_{k \cdot h}^{(k+1)h} \bar{\mathcal{L}}(\mathbf{x}, \dot{\mathbf{x}}^k) \, dt$$

Notice that the superscripts denote the variables associated with time, while the subscripts denote the variables associated with geometric position in this section.

To reduce the order of the discrete Lagrangian $\bar{\mathcal{L}}_d(\mathbf{x}^k, \dot{\mathbf{x}}^k)$, we take the velocity of discrete rod's vertices $\mathbf{v}^k = \left(\mathbf{v}_1^k, \mathbf{v}_2^k, \ldots, \mathbf{v}_{N_L}^k \right)$ as a new degree of freedom. Coupling the velocity $\mathbf{v}^k$ with geometric position of each vertices $\mathbf{x}^k = \left(\mathbf{x}_1^k, \mathbf{x}_2^k, \ldots, \mathbf{x}_{N_L}^k \right)$ and

$\mathbf{x}^{k-1}$ is realized through the constraints

$$\boldsymbol{g}^k = \boldsymbol{g}(\mathbf{x}^{k-1}, \mathbf{x}^k, \mathbf{v}^k) = \frac{(\mathbf{x}^k - \mathbf{x}^{k-1})}{h} - \mathbf{v}^k = 0 \tag{4.29}$$

Here $h \in \mathbb{R}$ is the time step, such that a continuous time period $[0, T]$ is discretized as $[0, h, 2h, 3h, \ldots, (N_t - 1)h, N_t h]$ ($N_t \in \mathbb{N}$ is the total number of time steps).

Then the discrete Lagrangian $\bar{\mathcal{L}}_d(\mathbf{x}^k, \dot{\mathbf{x}}^k)$ can be reformed as $\mathcal{L}_d(\mathbf{x}^k, \mathbf{v}^{k+1})$,

$$\mathcal{L}_d(\mathbf{x}^k, \mathbf{v}^{k+1}) = h\mathcal{L}(\mathbf{x}^k, \mathbf{v}^{k+1}) \approx \int_{k \cdot h}^{(k+1)h} \mathcal{L}(\mathbf{x}, \mathbf{v})\, dt$$

where

$$\mathcal{L}(\mathbf{x}^k, \mathbf{v}^{k+1}) = T - V \tag{4.30}$$

T is the kinetic energy $T = \frac{1}{2}\mathbf{v}^{k+1^T}\mathbf{M}\mathbf{v} + \mathbf{1}^k$; V is the potential energy $V = E(\Gamma(\mathbf{x}^k))$. Thus

$$\mathcal{L}_d(\mathbf{x}^k, \mathbf{v}^{k+1}) = h\left[\frac{1}{2}\mathbf{v}^{k+1^T}\mathbf{M}\mathbf{v}^{k+1} - E(\Gamma(\mathbf{x}^k))\right] \tag{4.31}$$

Next consider a discrete path of points $\left\{\mathbf{x}^k\right\}_{k=0}^{N_t}$ and calculate the discrete action along this sequence by summing the discrete Lagrangian on each time step, $S_d(\left\{\mathbf{x}^k\right\}) = \sum_{k=0}^{N_t} \mathcal{L}_d(\mathbf{x}^k, \mathbf{v}^{k+1})$. Following the continuous derivation above, we compute variations of this action sum with the boundary points $\mathbf{x}^0$, $\mathbf{x}^{N_t}$ held fixed. Combining the inextensibility constraints (4.18) and the constraints for coupling velocity and position (4.29), we can get

$$\begin{aligned}
\delta S_d(\{\mathbf{x}^k\}) &= \delta \sum_{k=0}^{N_t-1} \left[\mathcal{L}_d(\mathbf{x}^k, \mathbf{v}^{k+1}) + \mathbf{P}^{k+1} \cdot \boldsymbol{g}^{k+1} + \boldsymbol{\lambda}^{k+1} \cdot \Psi(\mathbf{x}^{k+1})\right] \\
&= \sum_{k=1}^{N_t-1} \left(\frac{\partial \mathcal{L}_d}{\partial \mathbf{x}^k} + \frac{\partial \boldsymbol{g}^k}{\partial \mathbf{x}^k}\mathbf{P}^k + \frac{\partial \boldsymbol{g}^{k+1}}{\partial \mathbf{x}^k}\mathbf{P}^{k+1} + \frac{\partial \Psi}{\partial \mathbf{x}^k}\boldsymbol{\lambda}^k\right) \cdot \delta \mathbf{x}^k \\
&+ \sum_{k=0}^{N_t-1} \left[\left(\frac{\partial \mathcal{L}_d}{\partial \mathbf{v}^{k+1}} + \frac{\partial \boldsymbol{g}^{k+1}}{\partial \mathbf{v}^{k+1}}\mathbf{P}^{k+1}\right) \cdot \delta \mathbf{v}^{k+1} + \boldsymbol{g}^{k+1} \cdot \delta \mathbf{P}^{k+1} + \Psi(\mathbf{x}^{k+1}) \cdot \delta \boldsymbol{\lambda}^{k+1}\right] \\
&+ \left(\frac{\partial \mathcal{L}_d}{\partial \mathbf{x}^0} + \frac{\partial \boldsymbol{g}^1}{\partial \mathbf{x}^0}\mathbf{P}^1\right) \cdot \delta \mathbf{x}^0 + \left(\frac{\partial \boldsymbol{g}^{N_t}}{\partial \mathbf{x}^{N_t}}\mathbf{P}^{N_t}\right) \cdot \delta \mathbf{x}^{N_t} = 0
\end{aligned}$$

Since the variations of the action be zero for any choice of $\delta \mathbf{x}^k, \delta \mathbf{v}^k, \delta P^k$, and $\delta \lambda^k$ with $\delta \mathbf{x}^0 = \delta \mathbf{x}^{N_t} = 0$, then we obtain the *constrained discrete Euler-Lagrange equations*,

$$\begin{aligned}
\frac{\partial \mathcal{L}_d}{\partial \mathbf{x}^k} + \frac{\partial \boldsymbol{g}^k}{\partial \mathbf{x}^k}\mathbf{P}^k + \frac{\partial \boldsymbol{g}^{k+1}}{\partial \mathbf{x}^k}\mathbf{P}^{k+1} + \frac{\partial \Psi}{\partial \mathbf{x}^k}\boldsymbol{\lambda}^k &= 0 \\
\frac{\partial \mathcal{L}_d}{\partial \mathbf{v}^{k+1}} + \frac{\partial \boldsymbol{g}^{k+1}}{\partial \mathbf{v}^{k+1}}\mathbf{P}^{k+1} &= 0 \\
\boldsymbol{g}^{k+1} = \boldsymbol{g}(\mathbf{x}^k, \mathbf{x}^{k+1}, \mathbf{v}^{k+1}) &= 0 \\
\Psi(\mathbf{x}^{k+1}) &= 0
\end{aligned} \tag{4.32}$$

Substituting equations (4.29) and (4.31) into equation (4.32), we can get,

$$\begin{aligned}
-h\frac{dE}{d\mathbf{x}^k} - \frac{\mathbf{P}^{k+1} - \mathbf{P}^k}{h} + \frac{\partial \Psi}{\partial \mathbf{x}^k}\boldsymbol{\lambda}^k &= 0 \\
h\mathbf{M}\mathbf{v}^{k+1} - \mathbf{P}^{k+1} &= 0 \\
(\mathbf{x}^{k+1} - \mathbf{x}^k) - h\mathbf{v}^{k+1} &= 0 \\
\Psi(\mathbf{x}^{k+1}) &= 0
\end{aligned} \tag{4.33}$$

The above equations can be further simplified by eliminating the terms $\mathbf{P}^{k+1}$ and $\mathbf{P}^k$. From the second equation in (4.33), we know $\mathbf{P}^{k+1} = h\mathbf{M}\mathbf{v}^{k+1}$ and $\mathbf{P}^k = h\mathbf{M}\mathbf{v}^k$. Substitute these two terms into the first equation in (4.33), we can get a new formed constrained discrete Euler-Lagrange equations,

$$\begin{cases} \mathbf{v}^{k+1} = \mathbf{v}^k + h\mathbf{M}^{-1}\left[-\dfrac{dE}{d\mathbf{x}^k} + \dfrac{1}{h}\dfrac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k\right] \\ \mathbf{x}^{k+1} = \mathbf{x}^k + h\mathbf{v}^{k+1} \\ \Psi(\mathbf{x}^{k+1}) = 0 \end{cases} \tag{4.34}$$

Assume there is a rigid object of mass m_o at speed $\mathbf{v}_b(t)$ under a constant load $\mathbf{F_c}$. It is well know that the speed of the object at time $t+\Delta t$ is $\mathbf{v}_b(t+\Delta t) = \mathbf{v}_b(t)+\Delta t\mathbf{F_c}/m_o$ according to the Newton's second law. Comparing this with the first equation in (4.34), we can reasonably consider the term $-\dfrac{dE}{d\mathbf{x}^k} + \dfrac{1}{h}\dfrac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k$ as the load applied at the vertices of discrete rod at time step $k+1$,

$$\mathbf{F}^{k+1} = -\frac{dE}{d\mathbf{x}^k} + \frac{1}{h}\frac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k \tag{4.35}$$

where

$$\mathbf{F}^{k+1}_{inext} = \frac{1}{h}\frac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k \tag{4.36}$$

can be taken as the force that guarantee the inextensibility of rod.

The conservative force $\mathbf{F}^e_i$ obtained through the elastic energy is[49],

$$\mathbf{F}^e_i = -\frac{d\mathbf{E}(\Gamma)}{d\mathbf{x}_i} = -\frac{\partial\mathbf{E}(\Gamma)}{\partial\mathbf{x}_i} - \frac{\partial\mathbf{E}(\Gamma)}{\partial\theta_{N_L}}\frac{\partial\theta_{N_L}}{\partial\mathbf{x}_i} \tag{4.37}$$

The term $-\dfrac{\partial\mathbf{E}(\Gamma)}{\partial\mathbf{x}_i}$ is contributed by the bending elastic energy and can be expressed as,

$$\begin{aligned}\frac{\partial \mathbf{E}(\Gamma)}{\partial \mathbf{x}_i} =& \frac{k_b}{\bar{l}_{i-1}} \frac{1}{(1+\mathbf{e}_{i-2}\cdot\mathbf{e}_{i-1})^2} \frac{\mathbf{x}_{i-1}-\mathbf{x}_{i-2}}{l_{i-2}\cdot l_{i-1}} \\ &+ \frac{k_b}{\bar{l}_i} \frac{1}{(1+\mathbf{e}_{i-1}\cdot\mathbf{e}_i)^2} \frac{\mathbf{x}_{i+1}-2\mathbf{x}_i+\mathbf{x}_{i-1}}{l_{i-1}\cdot l_i} \\ &- \frac{k_b}{\bar{l}_{i+1}} \frac{1}{(1+\mathbf{e}_i\cdot\mathbf{e}_{i+1})^2} \frac{\mathbf{x}_{i+2}-\mathbf{x}_{i+1}}{l_i\cdot l_{i+1}}\end{aligned}$$

The term $-\frac{\partial \mathbf{E}(\Gamma)}{\partial \theta_{N_L}} \frac{\partial \theta_{N_L}}{\partial \mathbf{x}_i}$ is contributed by the twisting elastic energy. Following the work by Bergou et al.[49], we know

$$-\frac{\partial \mathbf{E}(\Gamma)}{\partial \theta_{N_L}} \frac{\partial \theta_{N_L}}{\partial \mathbf{x}_i} = \frac{k_t(\theta_{N_L}-\theta_1)}{\bar{L}} \left(-\frac{(\mathbf{kb})_{i-1}}{2|\mathbf{e}_{i-1}|} - \frac{(\mathbf{kb})_i}{2|\mathbf{e}_{i-1}|} + \frac{(\mathbf{kb})_i}{2|\mathbf{e}_i|} + \frac{(\mathbf{kb})_{i+1}}{2|\mathbf{e}_i|} \right) \tag{4.38}$$

where we choose

$$(\mathbf{kb})_i = \mathbf{e}_{i-1} \times \mathbf{e}_i$$

To calculate the total twisting angle $\theta_{N_L} - \theta_1$, we adopt a new accumulative technique. The angular velocity of i_{th} segment can be expressed as [69]

$$\hat{\boldsymbol{\omega}}_i = \dot{\phi}_i \hat{\boldsymbol{n}}_i + \sin\phi \dot{\hat{\boldsymbol{n}}}_i + (1-\cos\phi_i)\hat{\boldsymbol{n}}_i \times \dot{\hat{\boldsymbol{n}}}_i \tag{4.39}$$

where $\hat{\boldsymbol{n}}_i = \frac{(\mathbf{kb})_i}{|(\mathbf{kb})_i|}$ is the unit vector along $(\mathbf{kb})_i$. Thus, the twisting angle change along each segment during each time step can be given by (details are given in the Appendix A)

$$\Delta\theta_i = h(\hat{\boldsymbol{\omega}}_i \cdot \mathbf{e}_i) = -\frac{h}{1+\mathbf{e}_{i-1}\cdot\mathbf{e}_i} (\mathbf{e}_{i-1}\times\mathbf{e}_i)\cdot(\dot{\mathbf{x}}_{i+1}-\dot{\mathbf{x}}_{i-1}) \tag{4.40}$$

Thus, the total twisting angle is

$$\theta_{N_L} - \theta_1 = \theta_{in} + h \sum_{i=2}^{N_L} \frac{(\mathbf{e}_{i-1} \times \mathbf{e}_i) \cdot (\dot{\mathbf{x}}_{i+1} - \dot{\mathbf{x}}_{i-1})}{1 + \mathbf{e}_{i-1} \cdot \mathbf{e}_i} \tag{4.41}$$

where θ_{in} is the input twisting angle for the rod.

Up to now, we have completely set up the time evolution of the discrete rod. Given $\mathbf{x}^k$ and $\mathbf{v}^k$ (time step k), we can employ some favorite nonlinear equation solver to find the unknowns $\mathbf{x}^{k+1}$, $\mathbf{v}^{k+1}$, and $\boldsymbol{\lambda}^k$ (time step $k+1$).

4.2.4 Picard iteration

To solve the discrete Euler-Lagrange equations (4.34), we employ Picard iteration. In the algorithm of Picard iteration, the inextensible constraints will be considered implicitly at the last step, so we can further simplify the equations (4.34) by eliminating the variable $\mathbf{v}^{k+1}$,

$$\mathbf{x}^{k+1} = \mathbf{x}^k + h\mathbf{v}^k + h^2\mathbf{M}^{-1}\left[-\frac{dE}{d\mathbf{x}^k} + \frac{1}{h}\frac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k\right] \tag{4.42}$$

Let $\bar{\mathbf{x}} = \mathbf{x}^k + h\mathbf{v}^k + h^2\mathbf{M}^{-1}\left(-\frac{dE}{d\mathbf{x}^k}\right)$, then the above equation becomes

$$\mathbf{x}^{k+1} = \bar{\mathbf{x}} + h^2\mathbf{M}^{-1}\left[\frac{1}{h}\frac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k\right] \tag{4.43}$$

Substituting the equation (4.18) into above equation, we can get,

$$\mathbf{x}_i^{k+1} = \bar{\mathbf{x}}_i + \frac{h^2}{m_i}\left(\frac{1}{h}(-2\lambda_i^k\mathbf{e}_i^k + 2\lambda_{i-1}^k\mathbf{e}_{i-1}^k)\right) \tag{4.44}$$

Let $\alpha_i = -\frac{2h\lambda_i}{m_i}$, such that

$$\mathbf{x}_i^{k+1} = \bar{\mathbf{x}}_i + \alpha_i^k \mathbf{e}_i^k - \alpha_{i-1}^k \mathbf{e}_{i-1}^k \tag{4.45}$$

Let $\bar{\mathbf{e}}_i = \bar{\mathbf{x}}_{i+1} - \bar{\mathbf{x}}_i$. Since $\mathbf{e}_i^{k+1} = \mathbf{x}_{i+1}^{k+1} - \mathbf{x}_i^{k+1}$, we can get,

$$\mathbf{e}_i^{k+1} = \bar{\mathbf{e}}_i + \alpha_{i+1}^k \mathbf{e}_{i+1}^k - 2\alpha_i^k \mathbf{e}_i^k + \alpha_{i-1}^k \mathbf{e}_{i-1}^k \tag{4.46}$$

Let $\underline{\boldsymbol{\alpha}}_i = [\alpha_{i-1}, \alpha_i, \alpha_{i+1}]^T$ and $\underline{\mathbf{e}}_i = [\mathbf{e}_{i-1}, -2\mathbf{e}_i, \mathbf{e}_{i+1}]$. The above equation can be reformed as

$$\mathbf{e}_i^{k+1} = \bar{\mathbf{e}}_i + \underline{\mathbf{e}}_i^k \underline{\boldsymbol{\alpha}}_i^k \tag{4.47}$$

Now considering the inextensible constraints (4.18), we know that $\mathbf{e}_i^{k+1}$ should satisfy

$$[\mathbf{e}_i^{k+1}]^T \mathbf{e}_i^{k+1} = l_i^2 \tag{4.48}$$

where l_i is the original length of the segment of discrete rod. Since

$$[\mathbf{e}_i^{k+1}]^T \mathbf{e}_i^{k+1} = [\bar{\mathbf{e}}_i]^T \bar{\mathbf{e}}_i + 2[\bar{\mathbf{e}}_i]^T \underline{\mathbf{e}}_i^k \underline{\boldsymbol{\alpha}}_i^k + [\underline{\boldsymbol{\alpha}}_i^k]^T [\underline{\mathbf{e}}_i^k]^T \underline{\mathbf{e}}_i^k \underline{\boldsymbol{\alpha}}_i^k$$

substitute equation (4.48) into the above equation, we know,

$$\left[2[\bar{\mathbf{e}}_i]^T \underline{\mathbf{e}}_i^k + [\underline{\boldsymbol{\alpha}}_i^k]^T [\underline{\mathbf{e}}_i^k]^T \underline{\mathbf{e}}_i^k\right] \underline{\boldsymbol{\alpha}}_i^k = l_i^2 - [\bar{\mathbf{e}}_i]^T \bar{\mathbf{e}}_i \tag{4.49}$$

Assemble the above equation, we can get,

$$[\mathbf{A} + \mathbf{B}(\boldsymbol{\alpha})]\boldsymbol{\alpha} = \mathbf{R} \tag{4.50}$$

where

$$A_{ij} = \begin{cases} [\bar{\mathbf{e}}_i]^T \mathbf{e}_{i-1}^k & \text{if } j = i-1 \\ -2[\bar{\mathbf{e}}_i]^T \mathbf{e}_i^k & \text{if } j = i \\ [\bar{\mathbf{e}}_i]^T \mathbf{e}_{i+1}^k & \text{if } j = i+1 \\ 0 & \text{if } j = others. \end{cases}$$

$$B_{ij} = \begin{cases} \left[\alpha_{i-1}(\mathbf{e}_{i-1}^k)^T - 2\alpha_i(\mathbf{e}_i^k)^T + \alpha_{i+1}(\mathbf{e}_{i+1}^k)^T\right] \mathbf{e}_{i-1}^k & \text{if } j = i-1 \\ -2\left[\alpha_{i-1}(\mathbf{e}_{i-1}^k)^T - 2\alpha_i(\mathbf{e}_i^k)^T + \alpha_{i+1}(\mathbf{e}_{i+1}^k)^T\right] \mathbf{e}_i^k & \text{if } j = i \\ \left[\alpha_{i-1}(\mathbf{e}_{i-1}^k)^T - 2\alpha_i(\mathbf{e}_i^k)^T + \alpha_{i+1}(\mathbf{e}_{i+1}^k)^T\right] \mathbf{e}_{i+1}^k & \text{if } j = i+1 \\ 0 & \text{if } j = others. \end{cases}$$

$$\mathbf{R} = \left[l_1^2 - [\bar{\mathbf{e}}_1]^T\bar{\mathbf{e}}_1, \;\; l_2^2 - [\bar{\mathbf{e}}_2]^T\bar{\mathbf{e}}_2, \;\; \ldots, \;\; l_i^2 - [\bar{\mathbf{e}}_i]^T\bar{\mathbf{e}}_i, \;\; \ldots, \;\; l_{N_L}^2 - [\bar{\mathbf{e}}_{N_L}]^T\bar{\mathbf{e}}_{N_L}\right]^T$$

To solve the equation (4.50), we first assume a value for $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{old}$ and substitute the value into $\mathbf{B}(\boldsymbol{\alpha})$; then the equation (4.50) becomes

$$[\mathbf{A} + \mathbf{B}(\boldsymbol{\alpha}_{old})]\boldsymbol{\alpha} = \mathbf{R}$$

Solving the above equation, we can get a new value for $\boldsymbol{\alpha} = \boldsymbol{\alpha}_{new}$. We assign the value of $\boldsymbol{\alpha}_{new}$ to $\boldsymbol{\alpha}_{old}$, substitute $\boldsymbol{\alpha}_{old}$ into the above equation, and solve the above equation again. Repeat the process until $\boldsymbol{\alpha}$ get convergent, then we get the solution for $\boldsymbol{\alpha}$.

Now the procedure for solving the discrete Euler-Lagrange equations by the algorithm of Picard iteration should be clear. We make a conclusion in the following,

i) Solve the equation (4.50) to obtain the value of $\boldsymbol{\alpha}$ using Picard iteration.

ii) Substitute $\boldsymbol{\alpha}$ into the equation (4.45) to update $\mathbf{x}^{k+1}$ the position information of vertices of discrete rod at the $k+1_{th}$ time step .

iii) Substitute $\mathbf{x}^{k+1}$ into equation (4.29) to update $\mathbf{v}^{k+1}$ the velocity of vertices at the $k+1_th$ time step.

iv) Repeat i), ii) and iii) until the final time step is reached.

4.3 Collision detection and response

The slenderness of the rod allows for very large bending and torsion leading to the possibility of self intersection. This is particularly true when tying knots, where the rod often collides with itself. This collision of portions of the rod with itself raises difficulties in collision detection and response [70]. To achieve real time simulation, efficient and precise detection and management of collisions are two keys to our model.

4.3.1 Collision detection

To detect the self-collisions, the intuitive method is to check the collisions between each pair segments of the rod. If the distance between two segments is the same as the distance between the two lines taken by the segments and is smaller than a certain value Δd, we define that the collision between these two segments happens (see Figure 4.3). The time complexity of this method is the square of the number of discrete rod vertices. Since the speed of the simulation is required to be at real time rate, the computation cost is unacceptable. There are a significant amount of fast collision detection techniques in the area of computer graphic to efficiently detect the possible collisions. Usually bounding volume hierarchy (BVH) and spatial partition are often employed to detection the self collisions of rod like one dimensional objects.

For the general BVH technique, the bounding volumes of each object forms the leaf nodes (see Figure 4.4). Then these nodes are grouped as small sets and enclosed

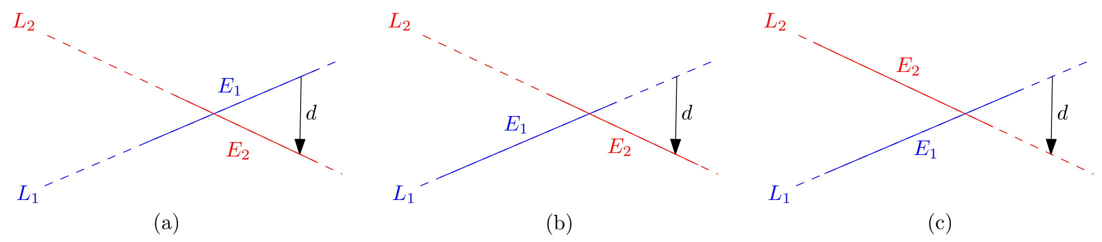


Figure 4.3: Line segments E_1 and E_2 are parts of lines L_1 and L_2 respectively. If the two ends of d, the distance between L_1 and L_2, locate on the line segments E_1 and E_2 and is smaller than a certain value Δd, we define that collision between E_1 and E_2 happens.

within larger bounding volumes. These are in turn grouped and enclosed within other larger bounding volumes in a recursive way, eventually resulting in a tree structure with a single bounding volume at the top. During the collision testing, children is not necessary to be checked if their parent bounding volume is not intersected. The time complexity can then be reduced to a logarithmic in the number of tests performed [71]. BVH is very efficient for the self collision detection of a rod in the average case [26]. However, constructing the hierarchy tree is essentially a recursive process and

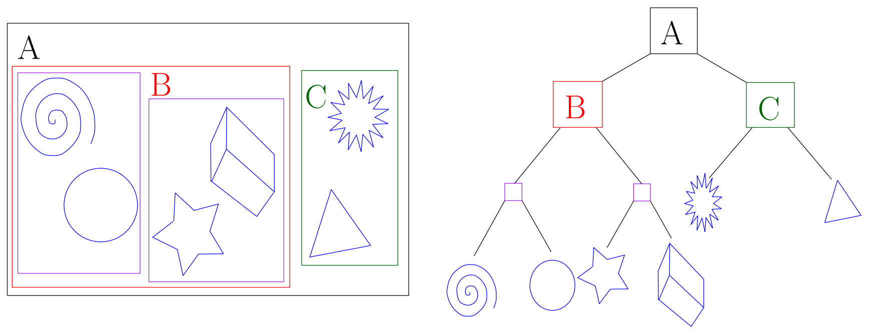


Figure 4.4: A sample bounding volume hierarchy of six objects.

thus is difficult to be parallelized for CPU especially for GPU processors, which will greatly limits the application when parallel computation are employed.

Spatial partitioning techniques provide broadphase processing by dividing space into regions and checking if objects intersect the same region of space. Because objects can only collide only if they overlap the same region of space, the number of pairwise test is significantly reduced. The common spatial partitioning methods include three major types, grids, trees, and spatial sorting. We will employ the uniform grids in our model, which is a highly parallelable and effective method employed to detect the self-collisions of a rod. This uniform grid divide space into a number of grid cells of equal size, as shown in Figure 4.5. Each object is then associated with the cells it overlaps. Possible collisions can only happen in the common and its neighbor cells.

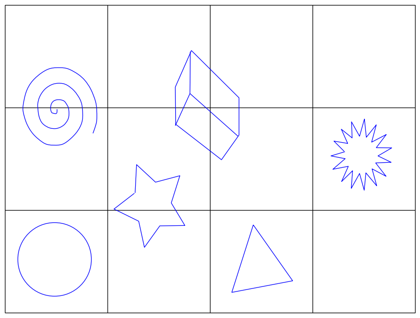


Figure 4.5: A sample spatial partitioning of six objects by uniform gird. The left bottom circle can only possible collide with the spiral, cuboid and the pentagram.

To obtain well performance of the spatial partitioning based on the uniform grid, the grid size should be carefully selected. If the grid size is too coarse, many objects may locate in the same grid cell, resulting in the expensive pairwise collision check. If the grid size is too fine, each object may take a lot of grid cells, and the computation cost of partitioning can be very expensive, which in turn slow down the collision check. In our case, the gird size should be larger than Δd (the criterion value judging the collision happens between two rod segment) to avoid missing detecting the possible self-collisions.

In our 3D cases, the gird cell we selected is a cubic. Let the edge length of the grid cell be ΔL_G. We then employ Bresenham's line algorithm [72], which is a very efficient algorithm major used to draw lines on monitors, to subdivide each rod segment into the gird cells. To better explain the Bresenham's line algorithm, we first explore a simple 2D case as shown in Figure 4.6. Consider a line with initial point (x_0, y_0) and end point (x_1, y_1). If $\Delta x = x_1 - x_0$ and $\Delta y = y_1 - y_0$, we define the driving axis to be the $x - axis$ if $|\Delta x| > |\Delta y|$ (the $y - axis$ if $|\Delta y| > |\Delta x|$). The driving axis is used as the âĂIJaxis of controlâĂİ for the algorithm and is the axis of maximum movement. The coordinate corresponding to the driving axis is incremented by one cell. The coordinate corresponding to the other axis (denoted as the passive axis) is only incremented as needed.

Bresenham's algorithm begins with the point (x_0, y_0) and âĂIJilluminatesâĂİ that pixel. Since we assume x is the driving axis in this example, it then increments the x coordinate by one cell of length l_{cell}. Rather than keeping track of the y coordinate (which increases by $m = \frac{\Delta y}{\Delta x} l_{cell}$, each time the x increases by one cell), the algorithm keeps an error ϵ at each stage, which represents the negative of the distance from the point where the line exits the cell to the middle of the cell. The calculation of error ϵ can be see in Figure 4.7,

$$\epsilon = y_0 + \frac{\Delta y}{\Delta x}(l_{cell} - x_0) - \frac{l_{cell}}{2} \tag{4.51}$$

If $\epsilon > l_{cell}$, then the cell position in the passive y axis increase 1 and let the new error to be

$$\epsilon = \epsilon - l_{cell}$$

When we consider the 3D cases, the procedure is exactly the same. Only difference is that we have two passive axis now (the detail algorithm is shown in Table 4.1).

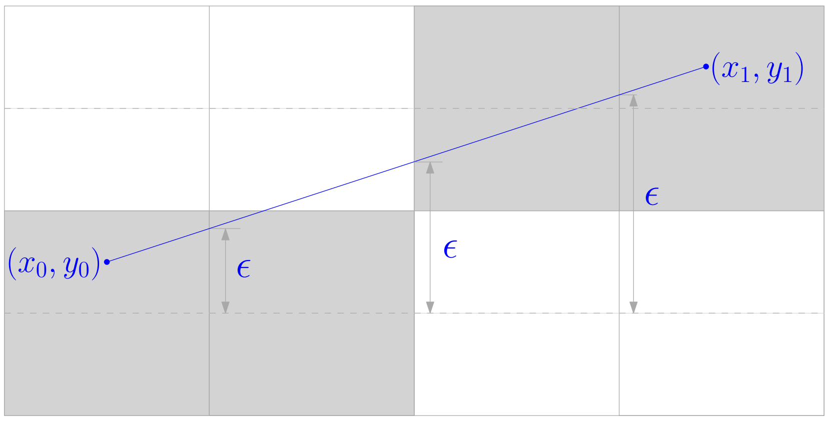


Figure 4.6: Illustration of the result of Bresenham's line algorithm.

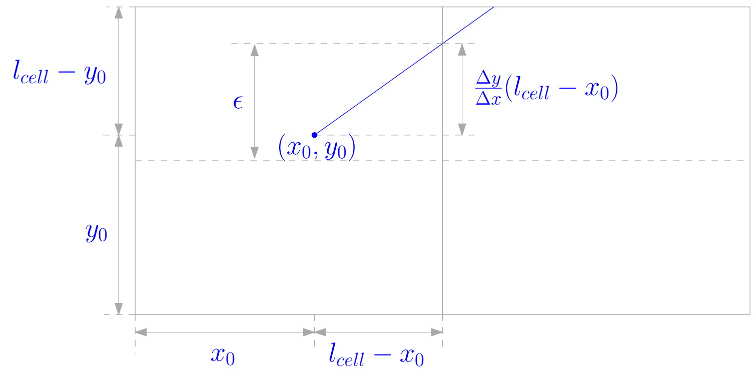


Figure 4.7: Illustration of the calculating the correct error ϵ of Bresenham's line algorithm.

```
1.  Let $\Delta x = x_1 - x_0$, $\Delta y = y_1 - y_0$, $\Delta z = z_1 - z_0$. Find the driving axis-
    $x - axis$ (assume $|\Delta x| = \max(|\Delta x|, |\Delta y|, |\Delta z|)$).
2.  Let $x_{inc} = (\Delta x < 0)? - 1 : 1$
        $y_{inc} = (\Delta y < 0)? - 1 : 1$
        $z_{inc} = (\Delta z < 0)? - 1 : 1$
3.  Calculate the initial bound error for the passive axis $\epsilon_y$ and $\epsilon_z$
    if $\Delta x > 0$,
      $\epsilon_y = 2|\Delta y| \left[1 - \left(\frac{x_0}{l_{cell}} - \lfloor\frac{x_0}{l_{cell}}\rfloor\right)\right] - y_{inc} \cdot |\Delta x| \left[1 - 2\left(\frac{y_0}{l_{cell}} - \lfloor\frac{y_0}{l_{cell}}\rfloor\right)\right]$
      $\epsilon_z = 2|\Delta z| \left[1 - \left(\frac{x_0}{l_{cell}} - \lfloor\frac{x_0}{l_{cell}}\rfloor\right)\right] - z_{inc} \cdot |\Delta x| \left[1 - 2\left(\frac{z_0}{l_{cell}} - \lfloor\frac{z_0}{l_{cell}}\rfloor\right)\right]$
    if $\Delta x < 0$,
      $\epsilon_y = y_{inc} \cdot 2|\Delta y| \left(\frac{x_0}{l_{cell}} - \lfloor\frac{x_0}{l_{cell}}\rfloor\right) - |\Delta x| \left[1 - 2\left(\frac{y_0}{l_{cell}} - \lfloor\frac{y_0}{l_{cell}}\rfloor\right)\right]$
      $\epsilon_z = z_{inc} \cdot 2|\Delta z| \left(\frac{x_0}{l_{cell}} - \lfloor\frac{x_0}{l_{cell}}\rfloor\right) - |\Delta x| \left[1 - 2\left(\frac{z_0}{l_{cell}} - \lfloor\frac{z_0}{l_{cell}}\rfloor\right)\right]$
4.  Let $P_x$, $P_y$, $P_z$ be the cell taken by the line segment.
5.  $P_x(0) = \lfloor\frac{x_0}{l_{cell}}\rfloor$, $P_y(0) = \lfloor\frac{y_0}{l_{cell}}\rfloor$, $P_z(0) = \lfloor\frac{z_0}{l_{cell}}\rfloor$.
    for $i = \lfloor\frac{x_0}{l_{cell}}\rfloor + 1$ to $\lfloor\frac{x_1}{l_{cell}}\rfloor$
        $P_x(i) = P_x(i-1) + x_{inc}$, $\epsilon_y = \epsilon_y + 2|\Delta y|$, $\epsilon_z = \epsilon_z + 2|\Delta z|$
        if $\epsilon_y > 2|\Delta x|$
            $\epsilon_y = \epsilon_y - 2\Delta x$, $P_y(i) = P_y(i-1) + y_{inc}$
        else
            $P_y(i) = P_y(i-1)$
        endif
        if $\epsilon_z > 2|\Delta x|$
            $\epsilon_z = \epsilon_z - 2\Delta x$, $P_z(i) = P_z(i-1) + z_{inc}$
        else
            $P_z(i) = P_z(i-1)$
        endif
    end for
```

Table 4.1: The application of Bresenham's algorithm to partition 3D line segment with start point (x_0, y_0, z_0) and end point (x_1, y_1, z_1) into a uniform grid of length l_{cell}.

4.3.2 Collision response

Once the collisions are detected, these collisions should be handled such that no penetrations between the discrete rod happen. At a broad level, collision response algorithms can be classified into three categories[73, 74]: constraint-based formulations[75, 76, 77], penalty-based methods[78], and impulse-based methods[79]. In general, constraint-based methods result in a more plausible simulation at the cost of extra computation. Impulse-based methods are more suitable for rigid body collision response. So we employ penalty-based method. However traditional penalty based methods suffer various kinds of problem, such as jitter effect. While Tang et al. improve the penalty method and develop a novel model-continuous penalty forces[73]. Therefore we employ the model of continuous penalty forces to handling the collision detection.

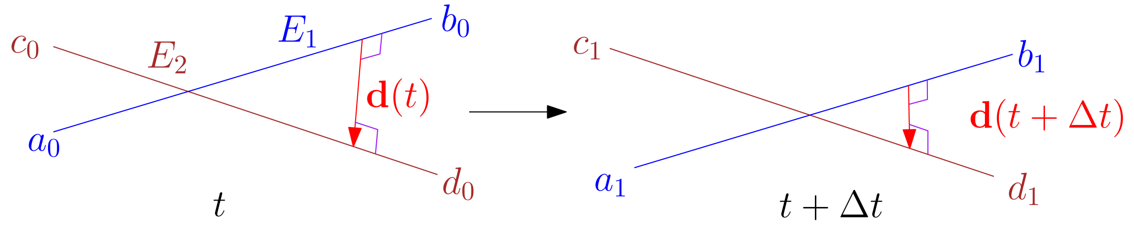


Figure 4.8: Continuous penalty force for edge-edge (EE) contact. $\mathbf{d}(t)$ is the displacement between the edge a_0b_0 (E_1) and edge c_0d_0 (E_2) at time t; $\mathbf{d}(t)$ is the distance between E_1 and E_2 at time $t + \Delta t$.

Since the rod is taken as one dimensional object and the radius of the cross section is neglect, we cannot handle the collision between the discrete straight edges as that for collision between 3D objects. As shown in figure 4.8.when the magnitude of the displacement $|\mathbf{d}(t)|$ between two straight edge E_1 and E_2 is less than Δd

($|\mathbf{d}(t)| \leq \Delta d$), the collision happens. Let the direction of the displacement $\mathbf{d}(t)$ is

$$\mathbf{U_d}(t) = \frac{\mathbf{d}(t)}{|\mathbf{d}(t)|} \tag{4.52}$$

According the penalty based method, if $|\mathbf{d}(t)| \leq \Delta d$, the penalty force created at time t is

$$\mathbf{F}_p(t) = k_s(\Delta d - |\mathbf{d}(t)|)\mathbf{U_d}(t) \tag{4.53}$$

where k_s is the stiffness constant. To derive our force formulation based on continuous penalty method, we pay attention to the impulse $\mathbf{I}$ produced by a penalty force $\mathbf{F}_p(t)$ during time instants $[t, t+\Delta t]$. The impulse $\mathbf{I}$ is defined using the following integral,

$$\mathbf{I} = \int_t^{t+\Delta t} \mathbf{F}_p(t)\ dt \tag{4.54}$$

According to the work by Tang et al.[73], the continuous penalty force created by the collision during the time interval Δt is

$$\mathbf{F}_c = \frac{\mathbf{I}}{\Delta t} = \frac{1}{\Delta t}\int_t^{t+\Delta t} \mathbf{F}_p(t)\ dt \tag{4.55}$$

The last task is to couple the continuous penalty force $\mathbf{F}_c$ into the constrained discrete Euler-Lagrange equation (4.34). Since our model to simulate the rod is force based model, we only need to insert $\mathbf{F}_c$ into the total load (4.35) of the equation (4.34). Therefore the collision combined constrained discrete Euler-Lagrange equation becomes

$$\begin{cases} \mathbf{v}^{k+1} = \mathbf{v}^k + h\mathbf{M}^{-1}\left[\left(-\dfrac{dE}{d\mathbf{x}^k} + \mathbf{F}_c\right) + \dfrac{1}{h}\dfrac{\partial\Psi}{\partial\mathbf{x}^k}\boldsymbol{\lambda}^k\right] \\ \mathbf{x}^{k+1} = \mathbf{x}^k + h\mathbf{v}^{k+1} \\ \Psi(\mathbf{x}^{k+1}) = 0 \end{cases} \tag{4.56}$$

We observe that to solve the collision combined constrained discrete Euler-Lagrange equations by Picard iteration, the only thing we need to do is to combine the continuous penalty force $\mathbf{F}_c$ together with the force $-\dfrac{dE}{d\mathbf{x}^k}$. Now we have finished to painted the whole picture of how to simulating the dynamic behavior of a rod.

4.4 Results

Before proceeding to the simulations results, we introduce a series of experiments to verify the validity of our rod model.

4.4.1 Qualitative experiment of plectoneme phenomenon

The Instron MicroTorsion machine (see Figure 4.10) was used to capture the formation of plectoneme when the thread was twisted. The specifications of the machine are shown in Table 4.2. Limited by the load measurement accuracy, we used a plastic wire (depicted in green in Figure 4.11) thicker and stiffer than surgical threads in the experiment. The schematic diagram of this qualitative experiments is shown in Figure 4.9. The left end of the thread is clamped to the input shaft that controls the twisting angle; the right end of the thread is clamped to the load cell that measures the torque applied on the thread.

The rest length of the sample is $L = 259$ mm and the distance between the input shaft and torque cell is $L_e = 164$ mm with the initial configuration shown in Figure 4.11a. We increase the twisting angle from 0 to 3600 deg at a rate of $V_\theta = 1$ deg/s.

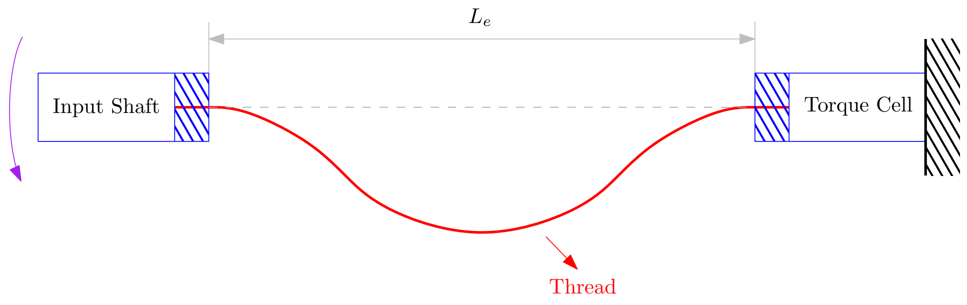


Figure 4.9: Digram of the experimental instruments. The thread is clamped at both ends on the input shaft (left) and torque cell (right). The twisting angle is controlled by the input shaft and the torque applied on the thread is measured through the torque cell. The distance between the input shaft and torque cell is fixed to L_e.

Figure 4.11b shows the final configuration of the wire (angle of twist equal to 3600 deg) with two loops formed (plectoneme).

Testing Speed Range	$0 \sim 60$ rpm
Resolution of Rotation	0.168 arc-min
Torque Cell Capacity	0.225 N $\cdot$ m
Load Measurement Accuracy	$\pm$ 0.5 of reading down to 1/500 of load cell capacity
Transducer Resolution	1 part in $500,000$ of $\pm$ full scale (19 bits)

Table 4.2: The specifications of Instron MicroTorsion Testing System MT2.

Figure 4.12 is a plot of torque applied on the thread with respect to angle of twist. We see two distinct deeps (points B and C). They correspond to the angle of twist at which loops have formed. An explanation for that behavior is that as the loop forms, twisting energy is converted into bending energy which results in a decrease in the amount of torque applied on the thread. When the loop has fully formed, self-contact occurs and prevents the thread from bending further. At the

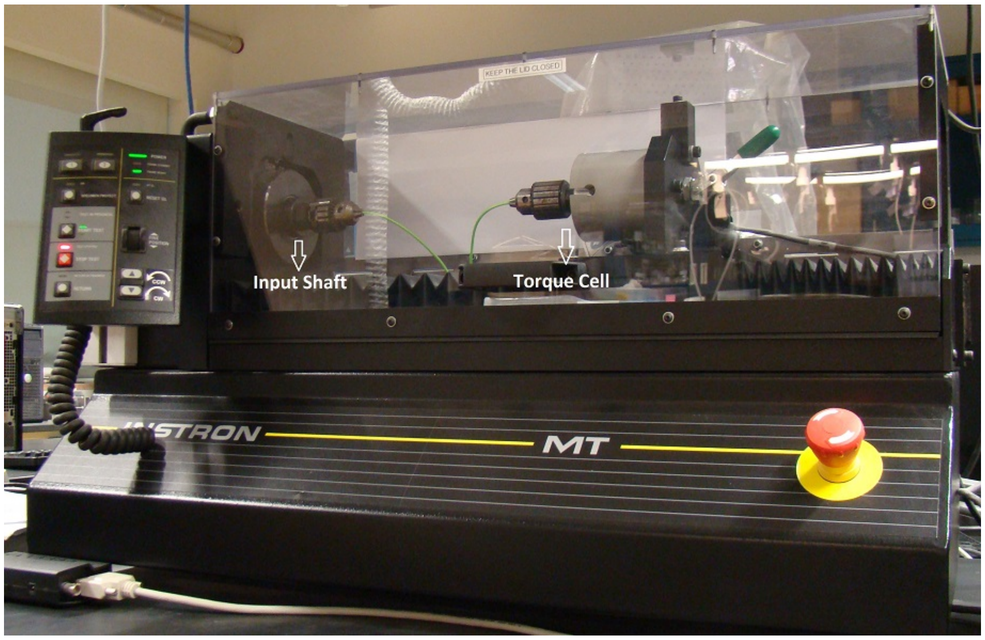
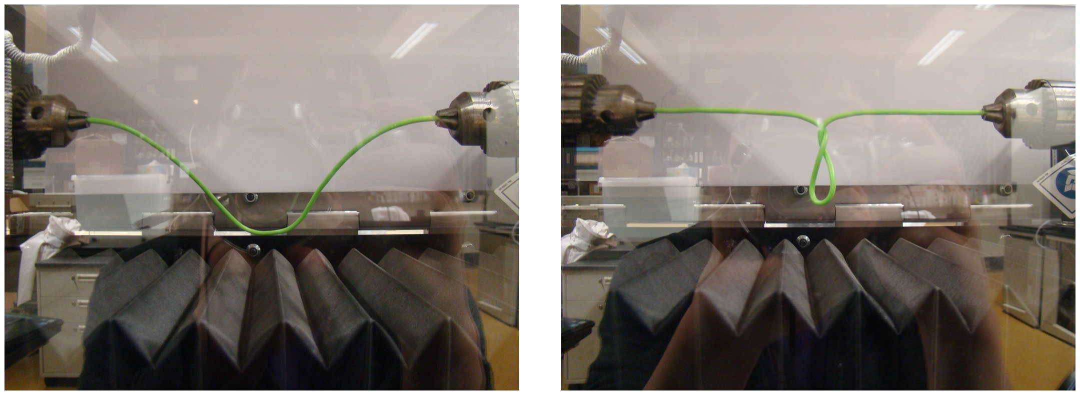


Figure 4.10: The picture of experimental instrument - Instron MicroTorsion Testing System MT2.



(a) Initial configuration of the thread (b) Final configuration of the thread

Figure 4.11: The configurations of the green plastic thread. The rest length of the thread is 259 mm. The distance between the two ends of the thread is $L_e = 164$ mm.

same time, twisting energy increases because of the input of angle of twist, which generates an increase of torque applied on the thread. In summary, the plectoneme phenomenon is associated with both a decrease and an increase of torque.

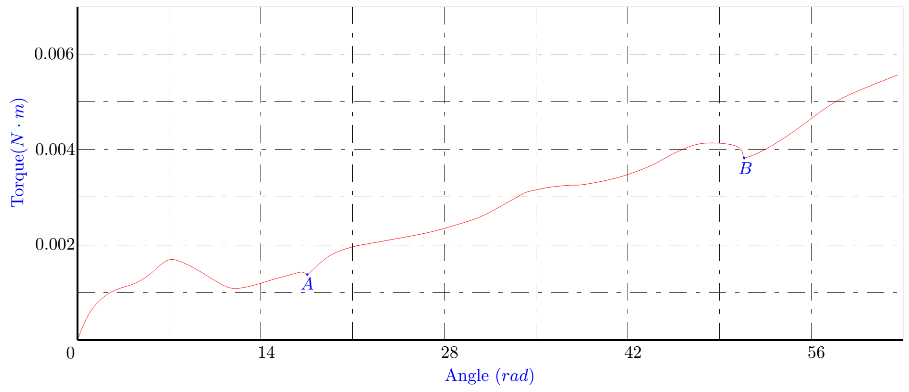


Figure 4.12: The plot of the torque applied on the thread with respect to angle of twist. Two sharp decrease points B and C correspond to the angle of twist at which loops of the thread have formed as shown in Figure 4.11b.

4.4.2 Simulation of plectoneme phenomenon

We apply the Kirchhoff rod model developed in this chapter to illustrate the formation of the plectoneme of a thread. Noteworthy parameters used in the simulations are a thread of length $L = 1$ m and mass $M = 0.02$ kg. The rod is discretized into $N_L = 20$ segments of equal length. The bending and torsional stiffness of the rod are $k_b = 0.02$ N·m^2 and $k_t = 0.02$ N·m^2 respectively. The time step is h = 0.001 s. For the collision detection and response, the distance criterion is $\Delta d = 0.3\frac{L}{N_L}$ and the stiffness constant is $k_s = 100$ N/m. The initial configuration of the rod is represented by a blue straight line on Figure 4.13.

In the simulation, the right end of the thread is displaced toward the left at a speed equal to $\mathbf{V}_e = [-0.5, 0, 0]^T$. When the distance between the two ends reaches the value of $L_e = 0.075$ m, the right end of the rod is twisted at an angular velocity equal to $\omega_e = 0.8$ rad/s until the angle of twist reaches $\theta_{in}^{\max} = 20$ rad (maximum

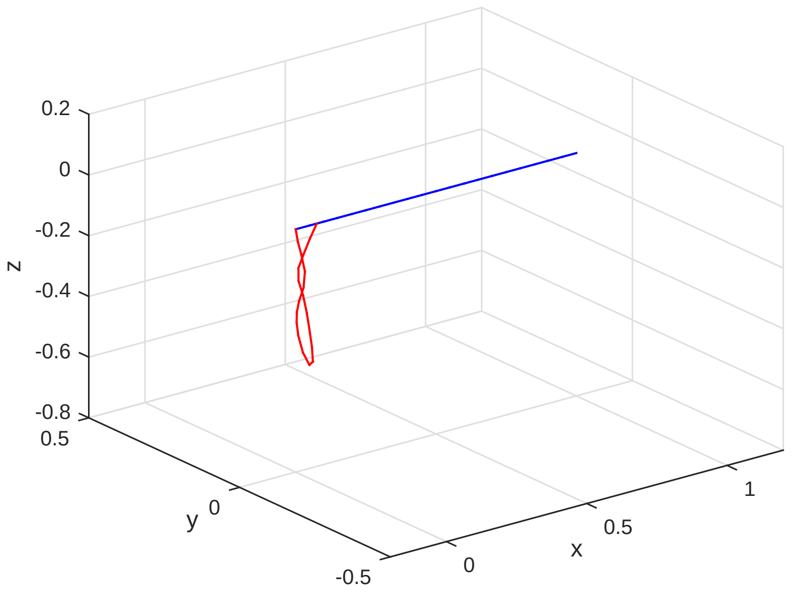


Figure 4.13: The left end of the thread is pinned and the right end is moving toward the left end at the speed of $\mathbf{V}_e = [-0.5, 0, 0]^T$ until the distance between the two ends is $L_e = 0.075$ m. Then the twisting angle is input with the angular velocity of $\omega_e = 0.8$ rad/s until reaching $\theta_{in}^{\max} = 25$ rad. The blue (straight) line represent the initial configuration of the thread at $t = 0.0$ s ; the red curve (helix) is the configuration at $t = 40.0$ s.

value). The simulations show that the formation of two loops is accompanied by a decrease then a increase of torque (points A and B in Figure 4.14), in agreement with the experimental results (points A and B in Figure 4.12). Also, the simulation shows some oscillations of the torque that are not recorded in the experiments. Due to the assumption of the elasticity of the thread model, the oscillations of the loop occurs and at the same time twisting energy and bending energy are keeping transfer to each other, resulting in the oscillations of the torque after the formation of the loops. Also, we should notice that, because part of the twisting energy has transfered to bending energy, the final torsion angle of the thread decrease and is smaller than

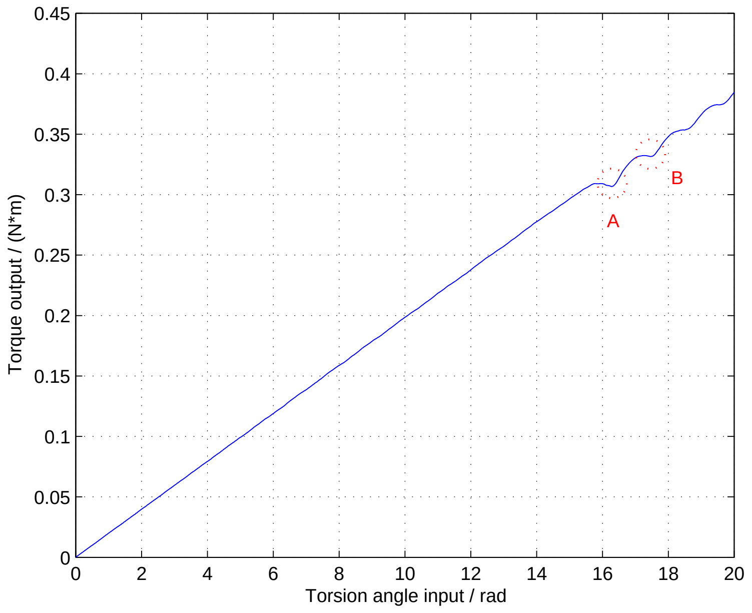


Figure 4.14: The relation between the torque on the thread and the input torsion angle during the simulation. Note that every contact causes an oscillation (the contact resulting in plectonemes are shown as points A and B), which shows a signature that is similar to the experimental results (see Figure 4.12).

the input torsion angle 20 rad (see Figure 4.15).

4.4.3 Real time simulation results

As we said, the goal of our project is to develop a physical based real time simulator, so we speed up the simulation by parallel programming techniques. Figure 4.16 shows the three phenomenons of plectoneme with different thread properties under the same load. The thread is initially straight with length $L = 1\ m$ and mass $M = 0.01$ kg, and discretized into $N = 100$ equal segments. The simulations are run with different stiffness shown in the Figures 4.16 (a) $k_b = 0.001$ N$\cdot$ m^2, $k_t = 0.001$ N$\cdot$ m^2, (b) $k_b = 0.0005$ N$\cdot$ m^2, $k_t = 0.001$ N$\cdot$ m^2, and (c) $k_b = 0.00025$ N$\cdot$ m^2, $k_t = 0.001$ N$\cdot$ m^2. During all the simulations, the time step is $h = 0.001$ s, and the

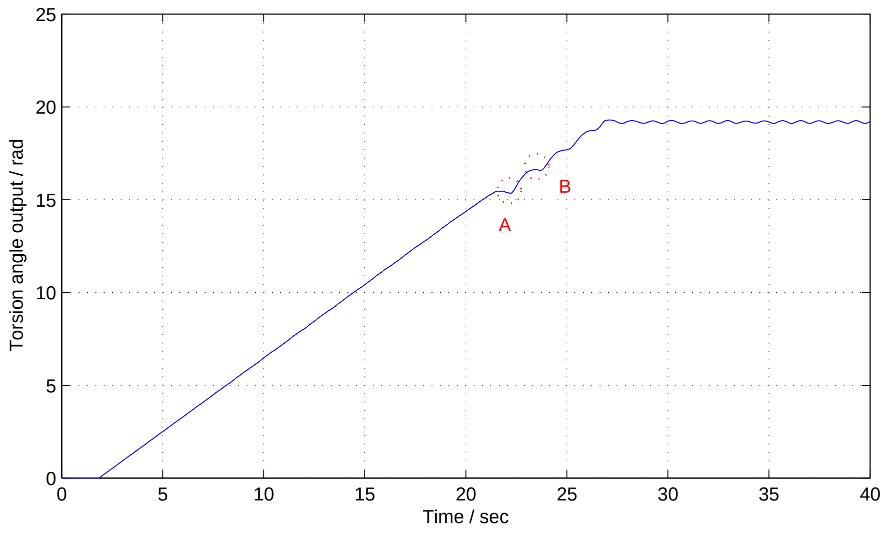
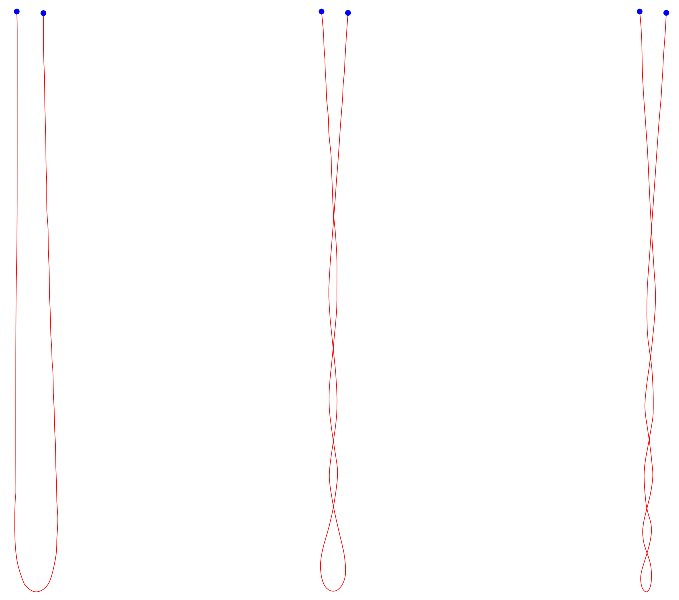


Figure 4.15: The figure shows the change of torsion angle of the thread with respect to time during the simulation. Note that every contact causes an oscillation (the contact resulting in plectonemes are shown as points A and B). The final torsion angle of the thread is smaller than the input torsion angle 20 rad.

same load is applied (the left end of the thread moves toward the fixed right end until the distance between the two ends is $L_e = 0.02$ m. Then the twisting angle is input with the angular velocity of $\omega_e = 20$ rad/s until reaching $\theta_{in}^{\max} = 50\pi$). We notice that with the same torsion stiffness, the thread with softer bending stiffness can generate more loops under the same torsional load. The simulation runs on a computer with Intel(R) Core(TM) i7-3520M CPU @ 2.90 GHz, 8.00 GB RAM, 64-bit Operation System (Win 8.1). The average frame rate of the graphical display remained above 1.00 KHz.

We also simulated the tying of a square knot. We kept all the previous parameters used for looping but discretized the rod into $N_L = 40$ segments of equal length. The result is shown in figure 4.17.



(a) $k_b = 0.001\ N \cdot m^2$
$k_t = 0.001\ N \cdot m^2$

(b) $k_b = 0.0005\ N \cdot m^2$
$k_t = 0.001\ N \cdot m^2$

(c) $k_b = 0.00025\ N \cdot m^2$
$k_t = 0.001\ N \cdot m^2$

Figure 4.16: The thread is initially straight with length $L = 1m$ and mass $M = 0.01kg$, and discretized into $N = 100$ equal segments. The simulations are run with different stiffness as shown in the figures (a), (b), and (c). During all the simulations, the time step is $h = 0.001$ s, and the same load is applied (the left end of the thread moves toward the fixed right end until the distance between the two ends is $L_e = 0.03m$. Then the twisting angle is input with the angular velocity of $\omega_e = 20$ rad/s until reaching $\theta_{in}^{\max} = 50\pi$). We note that for the same torsional modulus, the thread with the lower bending modulus has a more plectonemes (more loops).

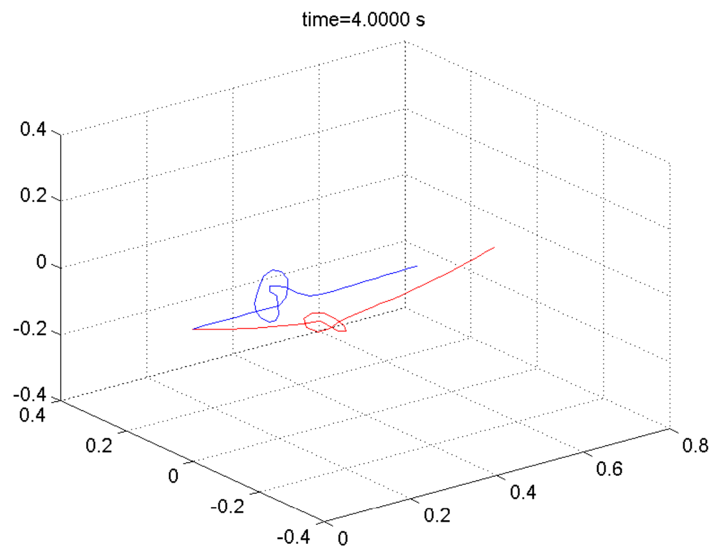


Figure 4.17: The result of tying a square knot. The left end of the thread is pinned and the right end is moving toward the right side. The blue (loosen) curve is the initial condition of the thread at $t = 0.0\ s$; the red (tight) curve is the configuration at $t = 4.0\ s$.

4.4.4 Application of the thread model to three dimensional scenario

The overarching goal of our project is to develop a surgery simulation software; therefore we need to verify that our thread model can serve as a physical engine of other application. In this dissertation, we test our thread model by apply it to a simple three dimensional scenario [*]. To apply our physical thread engine in various windows platform, we create a dynamic link library (DLL) file with the interface shown in Table 4.3. The results are shown in the Figure 4.18 and 4.19.

4.5 Conclusion

In this chapter, we develop an inextensible elastic rod model based on the theories of Kirchhoff rod, and this model can be used to simulate dynamic response

[*]This three dimensional scenario is generated by SimInsights Inc.

```
void Initialize(ThreadProperties &ThreadProp, double *InitDisX, int Loadtype); // Initialize the properties of the thread and select the load type (1. Motion input 2. End force input)
void ApplyMotionInput(double RightEndX, double RightEndY, double RightEndZ, double DeltaRightEndTwist, double LeftEndX, double LeftEndY, double LeftEndZ, double DeltaLeftEndTwist, double PenaltySpringConst); // Apply the motion input: the twist input angle (during each time step) and the position of the thread's two ends
void ApplyForceInput(double RightEndForceX, double RightEndForceY, double RightEndForceZ, double DeltaStartTwist, double LeftEndForceX, double LeftEndForceY, double LeftEndForceZ, double DeltaEndTwist); // Apply the force input: the twist input angle (during each time step) and the end force load applied on the thread's two ends
void Update(); // update the new position of the thread at each time step
void GetPosition(double *DisConfig); // get the position of the discrete thread at current time step
void Stop(); // stop the simulation
```

Table 4.3: The interfaces of the physical thread engine developed in this dissertation.

of a surgical thread at real time rate. The combination of maintaining exact inextensibility, together with the use of the discrete variational integrator technique and continuous collision force technique guarantees that the simulation is able to handle very high bending angles and intimate contact without instability. The symplectic properties of the variational integrator guarantee the conservation of momentum and energy, resulting in the long time stable simulation of the thread. To efficiently deal with the thread's possible self-collision when large deformation occurs, we employ fast collision detection based on uniform space portioning method and the collision response is managed through a continuous penalty forces. However, our model has not been coupled with tissue materials. Also, the thread is not allowed to apply large external forces. Additionally, real-time simulation of multiple threads may require faster computers.

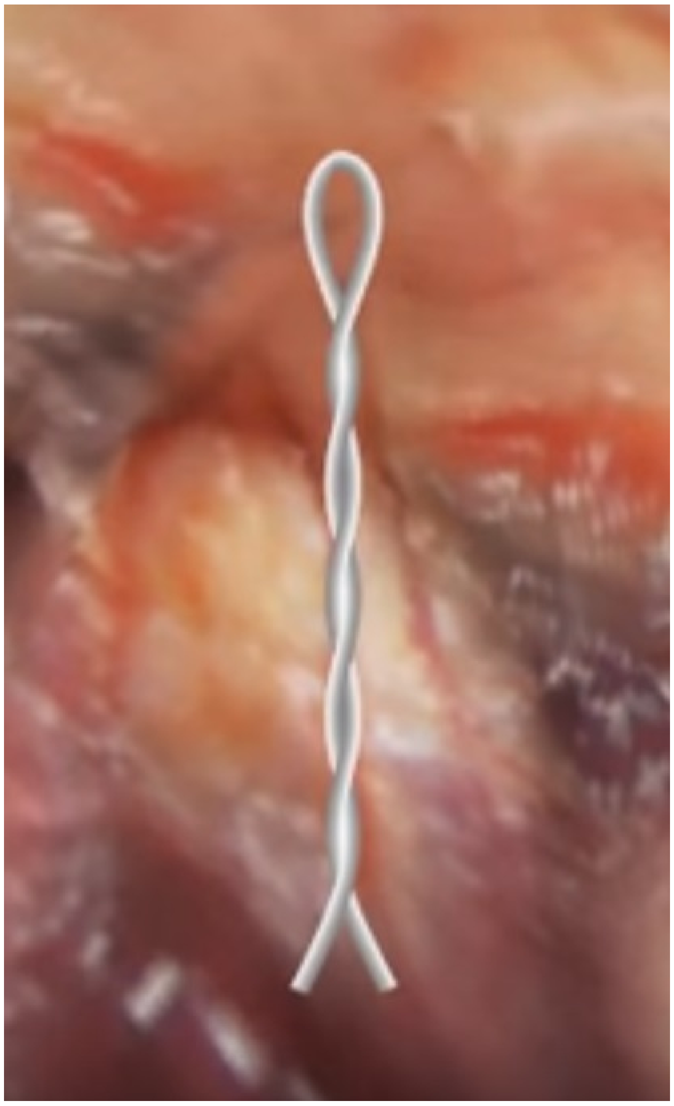
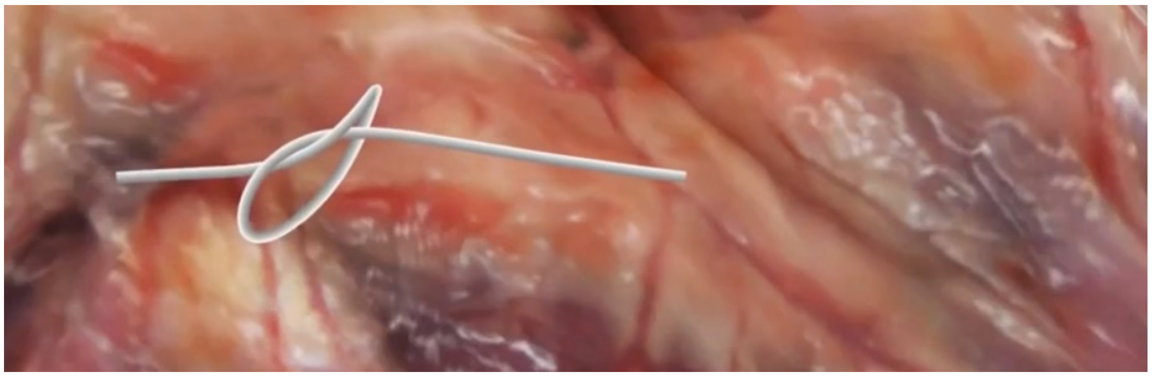


Figure 4.18: Plectoneme formation.


(a) Initial configuration of the knot.

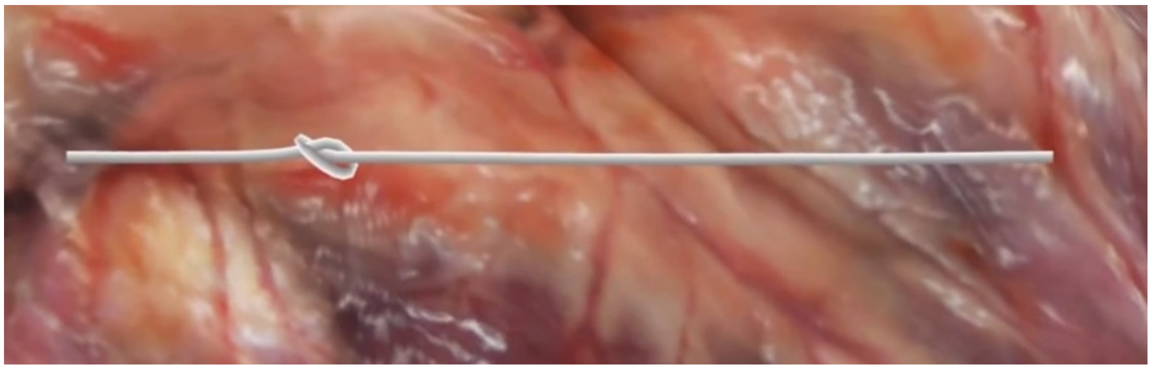

(b) Final configuration of the knot.

Figure 4.19: Knot tying.

REFERENCES

[1] Steven L Dawson. A critical approach to medical simulation. *Bulletin of the American college of Surgeons*, 87(11):12–18, 2002.

[2] Education Update. Modern surgery-minimally invasive and maximally restorative. `http://www.educationupdate.com/archives/2007/NOV/html/med-modern.html`, accessed on April 7, 2016.

[3] Johns Hopkins Medicine. Types of minimally invasive surgery. `http://www.hopkinsmedicine.org/minimally_invasive_robotic_surgery/types.html`, accessed on April 7, 2016.

[4] Rosh KV Sethi, Antonia J Henry, Nathanael D Hevelone, Stuart R Lipsitz, Michael Belkin, and Louis L Nguyen. Impact of hospital market competition on endovascular aneurysm repair adoption and outcomes. *Journal of Vascular Surgery*, 58(3):596–606, 2013.

[5] Center for Medical Simulation. Clinical training. `https://harvardmedsim.org/center-for-medical-simulation-clinical-training.php`, accessed on April 7, 2016.

[6] Physicions News. Robotic surgery at abington broadens its scope and success. `https://physiciansnews.com/2009/03/03/robotic-surgery-at-abington-broadens-its-scope-and-success/`, accessed on April 7, 2016.

[7] Intensive Care Unit. Echo-guided bi-caval dual lumen ecmo catheter insertion (avalon elite). `http://intensiveblog.com/`

echo-guided-avalon-ecmo-cannula-insertion/, accessed on April 7, 2016.

[8] Abbott Vascular. Indications and important safety information. http://www.abbottvascular.com/us/products/coronary-intervention/asahi-confianza.html, accessed on April 7, 2016.

[9] Nathan Hüsken. *Realtime Simulation of Stiff Threads for microsurgery training simulation.* PhD thesis, Ruperto-Carola University of Heidelberg, Germany, 2014.

[10] Andrew Nealen, Matthias Müller, Richard Keiser, Eddy Boxerman, and Mark Carlson. Physically based deformable models in computer graphics. In *Computer Graphics Forum*, volume 25, pages 809–836. Wiley Online Library, 2006.

[11] Simbionix Ltd. Basic and advanced suturing module. http://simbionix.com/simulators/lap-mentor/library-of-modules/basic-advanced-suturing/, accessed on April 7, 2016.

[12] V.I. Feodosyev. *Selected Problems and Questions in Strength of Materials.* Mir Publishers, 1977.

[13] G Domokos, P Holmes, and B Royce. Constrained Euler buckling. *Journal of Nonlinear Science*, 7(3):281–314, 1997.

[14] Herzl Chai. The post-buckling response of a bi-laterally constrained column. *Journal of the Mechanics and Physics of Solids*, 46(7):1155 – 1181, 1998.

[15] Philip Holmes, Gábor Domokos, John Schmitt, and Imre Szeberényi. Constrained Euler buckling: an interplay of computation and analysis. *Computer Methods in Applied Mechanics and Engineering*, 170(3):175–207, 1999.

[16] Jen-San Chen and Chia-Wei Li. Planar elastica inside a curved tube with clearance. *International Journal of Solids and Structures*, 44(18):6173–6186, 2007.

[17] Zhi-Hao Lu and Jen-San Chen. Deformations of a clamped–clamped elastica inside a circular channel with clearance. *International Journal of Solids and Structures*, 45(9):2470 – 2492, 2008.

[18] Srikrishna Doraiswamy, Krishna R Narayanan, and Arun R Srinivasa. Finding minimum energy configurations for constrained beam buckling problems using the Viterbi algorithm. *International Journal of Solids and Structures*, 49(2):289–297, 2012.

[19] Krishna R Narayanan and Arun R Srinivasa. Using discrete optimization algorithms to find minimum energy configurations of slender cantilever beams with non-convex energy functions. *Mechanics Research Communications*, 36(7):811–817, 2009.

[20] Clemson University. Spline curve - computer graphics course notes at Clemson University. `http://people.cs.clemson.edu/~dhouse/courses/405/notes/splines.pdf`, accessed on April 7, 2016.

[21] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. In *ACM Siggraph Computer Graphics*, volume 21, pages 205–214. ACM, 1987.

[22] Jeff Phillips, Andrew Ladd, and Lydia E Kavraki. Simulated knot tying. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 1, pages 841–846. IEEE, 2002.

[23] Julien Lenoir, Laurent Grisoni, and Christophe Chaillou. A suture model for surgical simulation. In *International Symposium on Medical Simulation*, pages

17–18, 2004.

[24] Julien Lenoir, Philippe Meseure, Laurent Grisoni, and Christophe Chaillou. Surgical thread simulation. In *ESAIM: Proceedings*, volume 12, pages 102–107. EDP Sciences, 2002.

[25] Adrien Theetten, Laurent Grisoni, Claude Andriot, and Brian Barsky. Geometrically exact dynamic splines. *Computer-Aided Design*, 40(1):35–48, 2008.

[26] Joel Brown, Jean-Claude Latombe, and Kevin Montgomery. Real-time knot-tying simulation. *The Visual Computer*, 20(2-3):165–179, 2004.

[27] Matthias Müller, Tae-Yong Kim, and Nuttapong Chentanez. Fast simulation of inextensible hair and fur. *VRIPHYS*, 12:39–44, 2012.

[28] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[29] Nobuyuki Umetani, Ryan Schmidt, and Jos Stam. Position-based elastic rods. In *ACM SIGGRAPH 2014 Talks*, page 47. ACM, 2014.

[30] Patricia Moore and Derek Molloy. A survey of computer-based deformable models. In *Machine Vision and Image Processing Conference, 2007. IMVIP 2007. International*, pages 55–66. IEEE, 2007.

[31] Robert E Rosenblum, Wayne E Carlson, and Edwin Tripp. Simulating the structure and dynamics of human hair: modelling, rendering and animation. *The Journal of Visualization and Computer Animation*, 2(4):141–148, 1991.

[32] Fei Wang, Etienne Burdet, Ankur Dhanik, Tim Poston, and Chee Leong Teo. Dynamic thread for real-time knot-tying. In *Eurohaptics Conference, 2005 and*

Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint, pages 507–508. IEEE, 2005.

[33] Fei Wang, Etienne Burdet, Vuillemin Ronald, and Hannes Bleuler. Knot-tying with visual and force feedback for vr laparoscopic training. In *2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pages 5778–5781. IEEE, 2005.

[34] Byoungwon Choe, Min Gyu Choi, and Hyeong-Seok Ko. Simulating complex hair with robust collision handling. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 153–160. ACM, 2005.

[35] Andrew Selle, Michael Lentine, and Ronald Fedkiw. A mass spring model for hair simulation. In *ACM Transactions on Graphics (TOG)*, volume 27, page 64. ACM, 2008.

[36] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 594–603. ACM, 2002.

[37] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behaviour. In *Graphics Interface*, pages 147–147. Canadian Information Processing Society, 1995.

[38] Blazej Kubiak, Nico Pietroni, Fabio Ganovelli, and Marco Fratarcangeli. A robust method for real-time thread simulation. In *Proceedings of the 2007 ACM Symposium on Rirtual Reality Software and Technology*, pages 85–88. ACM, 2007.

[39] Richard L Bishop. There is more than one way to frame a curve. *The American Mathematical Monthly*, 82(3):246–251, 1975.

[40] Ken-ichi Anjyo, Yoshiaki Usami, and Tsuneya Kurihara. A simple method for extracting the natural beauty of hair. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 111–120. ACM, 1992.

[41] Agnes Daldegan, Nadia Magnenat Thalmann, Tsuneya Kurihara, and Daniel Thalmann. An integrated system for modeling, animating and rendering hair. In *Computer Graphics Forum*, volume 12, pages 211–221. Wiley Online Library, 1993.

[42] Julien Lenoir, Stephane Cotin, Christian Duriez, and Paul Neumann. Interactive physically-based simulation of catheter and guidewire. *Computers & Graphics*, 30(3):416–422, 2006.

[43] Mordecai B Rubin. *Cosserat theories: shells, rods and points*, volume 79. Springer Science & Business Media, 2013.

[44] Dinesh K. Pai. STRANDS: interactive simulation of thin solids using Cosserat models. In *Computer Graphics Forum*, volume 21, pages 347–352. Wiley Online Library, 2002.

[45] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frédéric Leroy, and Jean-Luc Lévêque. Super-helices for predicting the dynamics of natural hair. *ACM Transactions on Graphics (TOG)*, 25(3):1180–1187, 2006.

[46] Jonas Spillmann and Matthias Teschner. CoRdE: Cosserat rod elements for the dynamic simulation of one-dimensional elastic objects. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 63–72. Eurographics Association, 2007.

[47] Jonas Spillmann and Matthias Teschner. An adaptive contact model for the robust simulation of knots. In *Computer Graphics Forum*, volume 27, pages 497–506. Wiley Online Library, 2008.

[48] Sukitti Punak and Sergei Kurenov. Simplified cosserat rod for interactive suture modeling. In *Medicine Meets Virtual Reality*, pages 466–472. IOS Press, 2011.

[49] Miklós Bergou, Max Wardetzky, Stephen Robinson, Basile Audoly, and Eitan Grinspun. Discrete elastic rods. In *ACM Transactions on Graphics (TOG)*, volume 27, page 63. ACM, 2008.

[50] EllisHarold Dill. Kirchhoff's theory of rods. *Archive for History of Exact Sciences*, 44(1):1–23, 1992.

[51] Joel Langer and David A Singer. Lagrangian aspects of the Kirchhoff elastic rod. *SIAM review*, 38(4):605–618, 1996.

[52] Maisheng Luo, Hongzhi Xie, Le Xie, Ping Cai, and Lixu Gu. A robust and real-time vascular intervention simulation based on kirchhoff elastic rod. *Computerized Medical Imaging and Graphics*, 38(8):735 – 743, 2014.

[53] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM Transactions on Graphics (TOG)*, 26(3):49, 2007.

[54] Wen Tang, Pierre Lagadec, Derek Gould, Tao Ruan Wan, Jianhua Zhai, and Thien How. A realistic elastic rod model for real-time simulation of minimally invasive vascular interventions. *The Visual Computer*, 26(9):1157–1165, 2010.

[55] Wen Tang, Tao Ruan Wan, Derek A Gould, Thien How, and Nigel W John. A stable and real-time nonlinear elastic approach to simulating guidewire and

catheter insertions based on cosserat rod. *Biomedical Engineering, IEEE Transactions on*, 59(8):2211–2218, 2012.

[56] D. Huang, W. Tang, T. R. Wan, N. W. John, D. Gould, Y. Ding, and Y. Chen. A new approach to haptic rendering of guidewires for use in minimally invasive surgical simulation. *Computer Animation and Virtual Worlds*, 22(2-3):261–268, 2011.

[57] Mireille Grégoire and Elmar Schömer. Interactive simulation of one-dimensional flexible parts. *Computer-Aided Design*, 39(8):694–707, 2007.

[58] Richard G Budynas. *Advanced strength and applied stress analysis.* McGraw-Hill Science/Engineering/Math, 1998.

[59] Richard Bellman. The theory of dynamic programming. *Bulletin of the American Mathematical Society*, 60(6):503–515, 1954.

[60] G David Forney Jr. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[61] Hubert Kaeslin. A gentle introduction to dynamic programming and the Viterbi algorithm. `http://www.cambridge.org/ar/download_file/160515/`, accessed on April 7, 2016.

[62] Nambirajan Seshadri and Carl-Erik W Sundberg. List Viterbi decoding algorithms with applications. *Communications, IEEE Transactions on*, 42(234):313–323, 1994.

[63] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[64] Sravani Nuti, Annie Ruimi, and JN Reddy. Modeling the dynamics of filaments for medical applications. *International Journal of Non-Linear Mechanics*, 66:139–148, 2014.

[65] Bruce R. Munson, Alric P. Rothmayer, Theodore H. Okiishi, and Wade W. Huebsch. *Fundamentals of Fluid Mechanics*. Wiley, 7 edition, 2012.

[66] Matthew West. *Variational integrators*. PhD thesis, California Institute of Technology, 2004.

[67] Jerrold E Marsden and Matthew West. Discrete mechanics and variational integrators. *Acta Numerica 2001*, 10:357–514, 2001.

[68] Liliya Kharevych, Weiwei Yang, Yiying Tong, Eva Kanso, Jerrold E Marsden, Peter Schröder, and Matthieu Desbrun. Geometric, variational integrators for computer animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 43–51. Eurographics Association, 2006.

[69] Malcolm D. Shuster. A survey of attitude representations. *Journal of the Astronautical Sciences*, 41(4):439 – 517, 1993.

[70] Yasuhiro Akutsu and Miki Wadati. Knots, links, braids and exactly solvable models in statistical mechanics. *Communications in Mathematical Physics*, 117(2):243–259, 1988.

[71] Christer Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology)*. Morgan Kaufmann Publishers Inc., 2004.

[72] Jack E Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[73] Min Tang, Dinesh Manocha, Miguel A Otaduy, and Ruofeng Tong. Continuous penalty forces. *ACM Trans. Graph.*, 31(4):107–1, 2012.

[74] Andrew Witkin. Physically based modeling: principles and practice constrained dynamics. *SIGGRAPH Course Notes*, pages 11–21, 1997.

[75] Robert Bridson, Ronald Fedkiw, and John Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 594–603. ACM, 2002.

[76] Christian Duriez, Frederic Dubois, Abderrahmane Kheddar, and Claude Andriot. Realistic haptic rendering of interacting deformable objects in virtual environments. *Visualization and Computer Graphics, IEEE Transactions on*, 12(1):36–47, 2006.

[77] Miguel A Otaduy, Rasmus Tamstorf, Denis Steinemann, and Markus Gross. Implicit contact handling for deformable objects. In *Computer Graphics Forum*, volume 28, pages 559–568. Wiley Online Library, 2009.

[78] Peter Wriggers, T Vu Van, and Erwin Stein. Finite element formulation of large deformation impact-contact problems with friction. *Computers & Structures*, 37(3):319–331, 1990.

[79] Brian Vincent Mirtich. *Impulse-based dynamic simulation of rigid body systems.* PhD thesis, University of California at Berkeley, 1996.

APPENDIX A

DERIVATION OF TOTAL TORSION ANGLE

To calculate the total twisting angle $\theta_{N_L} - \theta_1$, we adopt a new accumulative technique. The angular velocity of p_{th} segment can be expressed as [69]

$$\hat{\boldsymbol{\omega}}_p = \dot{\phi}_p \hat{\boldsymbol{n}}_p + \sin\phi \dot{\hat{\boldsymbol{n}}}_p + (1 - \cos\phi_p)\hat{\boldsymbol{n}}_p \times \dot{\hat{\boldsymbol{n}}}_p \tag{A.1}$$

where $\hat{\boldsymbol{n}}_p = \dfrac{(\mathbf{kb})_p}{|(\mathbf{kb})_p|}$ is the unit vector along $(\mathbf{kb})_p$, as shown in Figure A.1. Since $(\mathbf{kb})_p = \mathbf{e}_w \times \mathbf{e}_p$, the magnitude of $|(\mathbf{kb})_p|$ is $\sin\phi_p$ ($|(\mathbf{kb})_p| = \sin\phi_p$), where ϕ_p is the angle between two continuous discrete segments, and

$$(\mathbf{kb})_p = \sin\phi_p \hat{\boldsymbol{n}}_p$$

$$\dot{(\mathbf{kb})}_p = \dot{\mathbf{e}}_w \times \mathbf{e}_p + \mathbf{e}_w \times \dot{\mathbf{e}}_p$$

Therefore

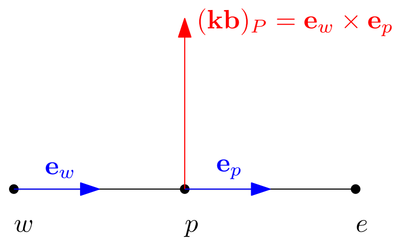


Figure A.1: The solution space of all the possible paths.

$$\begin{aligned}\dot{\hat{\boldsymbol{n}}}_p &= -\frac{\cos\phi_p}{\sin^2\phi_p}(\mathbf{kb})_p + \frac{1}{\sin\phi_p}(\dot{\mathbf{kb}})_p \\ &= -\frac{\cos\phi_p}{\sin^2\phi_p}\left(\mathbf{e}_w \times \mathbf{e}_p\right) + \frac{1}{\sin\phi_p}\left(\dot{\mathbf{e}}_w \times \mathbf{e}_p + \mathbf{e}_w \times \dot{\mathbf{e}}_p\right)\end{aligned} \tag{A.2}$$

Thus, the twisting angle change along each segment during each time step can be given by,

$$\begin{aligned}\Delta\theta_p &= h(\hat{\boldsymbol{\omega}}_p \cdot \mathbf{e}_p) = h(\mathbf{e}_p \cdot \hat{\boldsymbol{\omega}}_p) \\ &= h\left[\dot{\phi}_p\ \mathbf{e}_p \cdot \hat{\boldsymbol{n}}_p + \sin\phi\ \mathbf{e}_p \cdot \dot{\hat{\boldsymbol{n}}}_p + (1-\cos\phi_p)\ \mathbf{e}_p \cdot (\hat{\boldsymbol{n}}_p \times \dot{\hat{\boldsymbol{n}}}_p)\right]\end{aligned} \tag{A.3}$$

The first term in above equation A.3 is

$$\begin{aligned}\dot{\phi}_p\ \mathbf{e}_p \cdot \hat{\boldsymbol{n}}_p &= \dot{\phi}_p\ \mathbf{e}_p \cdot \frac{\mathbf{kb}_p}{\sin\phi_p} \\ &= \frac{\dot{\phi}_p}{\sin\phi_p}\mathbf{e}_p \cdot (\mathbf{e}_w \times \mathbf{e}_p) = 0\end{aligned} \tag{A.4}$$

The second term in the equation A.3 is

$$\begin{aligned}\sin\phi\ \mathbf{e}_p \cdot \dot{\hat{\boldsymbol{n}}}_p &= \sin\phi\ \mathbf{e}_p \cdot \left[-\frac{\cos\phi_p}{\sin^2\phi_p}(\mathbf{kb})_p + \frac{1}{\sin\phi_p}(\dot{\mathbf{kb}})_p\right] \\ &= -\frac{\cos\phi_p}{\sin\phi_p}\ \mathbf{e}_p \cdot \mathbf{kb}_p + \mathbf{e}_p \cdot (\dot{\mathbf{kb}})_p\end{aligned} \tag{A.5}$$

From equation A.4, we know the first term in above equation is 0. Thus,

$$\begin{aligned}\sin\phi\ \mathbf{e}_p \cdot \dot{\hat{\boldsymbol{n}}}_p &= 0 + \mathbf{e}_p \cdot (\dot{\mathbf{kb}})_p = \mathbf{e}_p \cdot [\dot{\mathbf{e}}_w \times \mathbf{e}_p + \mathbf{e}_w \times \dot{\mathbf{e}}_p] \\ &= \mathbf{e}_p \cdot [\dot{\mathbf{e}}_w \times \mathbf{e}_p] + \mathbf{e}_p \cdot [\mathbf{e}_w \times \dot{\mathbf{e}}_p] \\ &= \dot{\mathbf{e}}_w \cdot [\mathbf{e}_p \times \mathbf{e}_p] + \dot{\mathbf{e}}_p \cdot [\mathbf{e}_p \times \mathbf{e}_w] \\ &= \dot{\mathbf{e}}_p \cdot [\mathbf{e}_p \times \mathbf{e}_w] = -\dot{\mathbf{e}}_p \cdot [\mathbf{e}_w \times \mathbf{e}_p] \\ &= -(\mathbf{kb})_p \cdot \dot{\mathbf{e}}_p\end{aligned} \tag{A.6}$$

The third term in equation A.3 is

$$
\begin{aligned}
&(1-\cos\phi_p)\ \mathbf{e}_p\cdot(\hat{\boldsymbol{n}}_p\times\dot{\hat{\boldsymbol{n}}}_p)\\
&=(1-\cos\phi_p)\ \mathbf{e}_p\cdot\left[\frac{1}{\sin\phi_p}(\mathbf{kb})_p\times\left(-\frac{\cos\phi_p}{\sin^2\phi_p}(\mathbf{kb})_p+\frac{1}{\sin\phi_p}\dot{(\mathbf{kb})}_p\right)\right]\\
&=-\frac{(1-\cos\phi_p)\cos\phi_p}{\sin^3\phi_p}\mathbf{e}_p\cdot[(\mathbf{kb})_p\times(\mathbf{kb})_p]+\frac{1-\cos\phi_p}{\sin^2\phi_p}\mathbf{e}_p\cdot\left[(\mathbf{kb})_p\times\dot{(\mathbf{kb})}_p\right]\\
&=\frac{1-\cos\phi_p}{1-\cos^2\phi_p}\mathbf{e}_p\cdot[(\mathbf{kb})_p\times(\dot{\mathbf{e}}_w\times\mathbf{e}_p+\mathbf{e}_w\times\dot{\mathbf{e}}_p)]\\
&=\frac{1-\cos\phi_p}{(1-\cos\phi_p)(1+\cos\phi_p)}\mathbf{e}_p\cdot[(\mathbf{kb})_p\times(\dot{\mathbf{e}}_w\times\mathbf{e}_p)+(\mathbf{kb})_p\times(\mathbf{e}_w\times\dot{\mathbf{e}}_p)]\\
&=\frac{1}{1+\cos\phi_p}\mathbf{e}_p\cdot\{\mathbf{e}_w\,[(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_p]-\dot{\mathbf{e}}_p\,[(\mathbf{kb})_p\cdot\mathbf{e}_w]+\dot{\mathbf{e}}_w\,[(\mathbf{kb})_p\cdot\mathbf{e}_p]-\mathbf{e}_p\,[(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_w]\}\\
&=\frac{1}{1+\cos\phi_p}\,\{\mathbf{e}_p\cdot\mathbf{e}_w\,[(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_p]-\mathbf{e}_p\cdot\mathbf{e}_p\,[(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_w]\}\\
&=\frac{1}{1+\cos\phi_p}\,[\cos\phi_p(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_p-(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_w]\\
&=\frac{1}{1+\cos\phi_p}(\mathbf{kb})_p\cdot(\cos\phi_p\dot{\mathbf{e}}_p-\dot{\mathbf{e}}_w)
\end{aligned}
\tag{A.7}
$$

Substituting equations (A.4), (A.6), and (A.7) into equation (A.3), we can obtain

$$
\begin{aligned}
\Delta\theta_p&=h\left[\dot{\phi}_p\ \mathbf{e}_p\cdot\hat{\boldsymbol{n}}_p+\sin\phi\ \mathbf{e}_p\cdot\dot{\hat{\boldsymbol{n}}}_p+(1-\cos\phi_p)\ \mathbf{e}_p\cdot(\hat{\boldsymbol{n}}_p\times\dot{\hat{\boldsymbol{n}}}_p)\right]\\
&=h\left[-(\mathbf{kb})_p\cdot\dot{\mathbf{e}}_p+\frac{1}{1+\cos\phi_p}(\mathbf{kb})_p\cdot(\cos\phi_p\dot{\mathbf{e}}_p-\dot{\mathbf{e}}_w)\right]\\
&=h(\mathbf{kb})_p\cdot\frac{\cos\phi_p\dot{\mathbf{e}}_p-\dot{\mathbf{e}}_w+(1+\cos\phi_p)\dot{\mathbf{e}}_w}{1+\cos\phi_p}\\
&=-\frac{h}{1+\cos\phi_p}(\mathbf{kb})_p\cdot(\dot{\mathbf{e}}_w+\dot{\mathbf{e}}_p)=-\frac{h}{1+\cos\phi_p}(\mathbf{kb})_p\cdot(\dot{\mathbf{x}}_e-\dot{\mathbf{x}}_w)\\
&=-\frac{h}{1+\cos\phi_p}(\mathbf{e}_w\times\mathbf{e}_p)\cdot(\dot{\mathbf{x}}_e-\dot{\mathbf{x}}_w)
\end{aligned}
\tag{A.8}
$$

which is the equation (4.40).

APPENDIX B

DERIVATION OF CONTINOUS PENALTY FORCES

Our method to handle the collision response is based on the model of continuous penalty force is developed by Tang et. al. [73]; hence our notation here follows their work. Since we are going to develop a model that is able to provide force feedback, the time step is very small (the simulation rate is at least 600 Hz). As shown in Figure B.1, we assume that two line segments $\mathbf{L}_1(\alpha) = \mathbf{P} + \alpha\mathbf{d}_1$ ($\mathbf{d}_1 = \mathbf{Q} - \mathbf{P}$, $\alpha \in [0, 1]$) and $\mathbf{L}_2(\beta) = \mathbf{R} + \beta\mathbf{d}_2$ ($\mathbf{d}_2 = \mathbf{S} - \mathbf{R}$, $\beta \in [0, 1]$) collide with each other ($|\mathbf{w}_c| < \Delta d$), where the shortest displacement between them is $\mathbf{w}_c$ with start point $\mathbf{L}_2(\beta_c)$ on $\mathbf{L}_2$ and end point $\mathbf{L}_1(\alpha_c)$ on $\mathbf{L}_1$. The motion of the four ends points of the two line segments are given by

$$\mathbf{P}(t) = \mathbf{P}_0 + \mathbf{V}_P t \qquad \mathbf{Q}(t) = \mathbf{Q}_0 + \mathbf{V}_Q t$$
$$\mathbf{R}(t) = \mathbf{R}_0 + \mathbf{V}_R t \qquad \mathbf{S}(t) = \mathbf{S}_0 + \mathbf{V}_S t$$

Let $a = \mathbf{d}_1 \cdot \mathbf{d}_1$, $b = \mathbf{d}_1 \cdot \mathbf{d}_2$, $c = \mathbf{d}_2 \cdot \mathbf{d}_2$, $d = \mathbf{d}_1 \cdot \mathbf{w}$, $e = \mathbf{d}_2 \cdot \mathbf{w}$, where $\mathbf{w} = \mathbf{P} - \mathbf{R}$, then we can obtain [73]

$$\alpha_c(t) = \frac{be - cd}{ac - b^2}, \qquad \beta_c(t) = \frac{ae - bd}{ac - b^2} \tag{B.1}$$

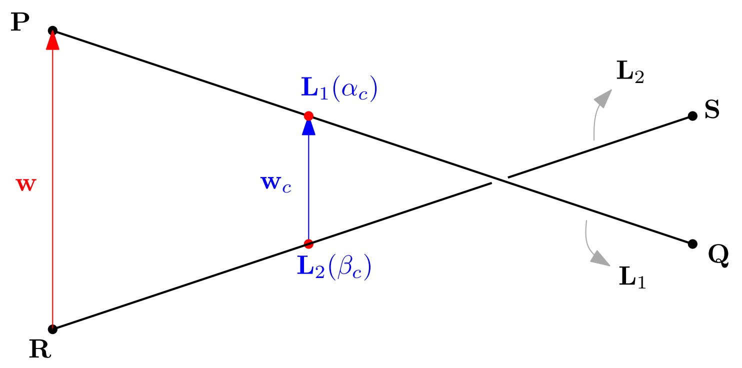


Figure B.1: Two line segments $\mathbf{L}_1(\alpha) = \mathbf{P} + \alpha\mathbf{d}_1$ ($\mathbf{d}_1 = \mathbf{Q} - \mathbf{P}$, $\alpha \in [0,1]$) and $\mathbf{L}_2(\beta) = \mathbf{R} + \beta\mathbf{d}_2$ ($\mathbf{d}_2 = \mathbf{S} - \mathbf{R}$, $\beta \in [0,1]$) collide with each other ($|\mathbf{w}_c| < \Delta d$), where the shortest displacement between them is $\mathbf{w}_c$ with start point $\mathbf{L}_2(\beta_c)$ on $\mathbf{L}_2$ and end point $\mathbf{L}_1(\alpha_c)$ on $\mathbf{L}_1$.

Therefore we know

$$\begin{aligned}
\mathbf{w}_c(t) &= \mathbf{L}_1(\alpha_c) - \mathbf{L}_2(\beta_c) \\
&= \mathbf{w} + \frac{(be-ac)\mathbf{d}_1 - (ae-bd)\mathbf{d}_2}{ac-b^2} \\
&= \mathbf{P} - \mathbf{R} + \frac{(be-ac)(\mathbf{Q}-\mathbf{P}) - (ae-bd)(\mathbf{S}-\mathbf{R})}{ac-b^2} \\
&= \left(1 - \frac{be-cd}{ac-b^2}\right)\mathbf{P} + \left(\frac{be-cd}{ac-b^2}\right)\mathbf{Q} - \left(1 - \frac{ae-bd}{ac-b^2}\right)\mathbf{R} - \frac{ae-bd}{ac-b^2}\mathbf{S} \\
&= \omega_P(t)\mathbf{P}(t) + \omega_Q(t)\mathbf{Q}(t) - \omega_R(t)\mathbf{R}(t) - \omega_S(t)\mathbf{S}(t)
\end{aligned} \tag{B.2}$$

where $\omega_P(t) = 1 - \frac{be-cd}{ac-b^2}$, $\omega_Q(t) = \frac{be-cd}{ac-b^2}$, $\omega_R(t) = 1 - \frac{ae-bd}{ac-b^2}$, and $\omega_S(t) = \frac{ae-bd}{ac-b^2}$. When $ac - b^2 = 0$, $\mathbf{L}_1$ and $\mathbf{L}_2$ are parallel to each other. The direction of

$\mathbf{w}_c(t)$ can be given by,

$$\begin{aligned}
\mathbf{m}'(t) &= (\mathbf{Q}(t) - \mathbf{P}(t)) \times (\mathbf{R}(t) - \mathbf{S}(t)) \\
&= [(\mathbf{Q}_0 - \mathbf{P}_0) + (\mathbf{V}_Q - \mathbf{V}_P)t] \times [(\mathbf{R}_0 - \mathbf{S}_0) + (\mathbf{V}_R - \mathbf{V}_S)t] \\
&= (\mathbf{Q}_0 - \mathbf{P}_0) \times (\mathbf{R}_0 - \mathbf{S}_0) + [(\mathbf{V}_Q - \mathbf{V}_P) \times (\mathbf{R}_0 - \mathbf{S}_0) \\
&+ (\mathbf{Q}_0 - \mathbf{P}_0) \times (\mathbf{V}_R - \mathbf{V}_S)]\, t + (\mathbf{V}_Q - \mathbf{V}_P) \times (\mathbf{V}_R - \mathbf{V}_S)t^2 \\
&= \mathbf{n}'_0 + \mathbf{n}'_1 t + \mathbf{n}'_2 t^2
\end{aligned} \tag{B.3}$$

where

$$\begin{aligned}
\mathbf{n}'_0 &= (\mathbf{Q}_0 - \mathbf{P}_0) \times (\mathbf{R}_0 - \mathbf{S}_0) \\
\mathbf{n}'_1 &= (\mathbf{V}_Q - \mathbf{V}_P) \times (\mathbf{R}_0 - \mathbf{S}_0) + (\mathbf{Q}_0 - \mathbf{P}_0) \times (\mathbf{V}_R - \mathbf{V}_S) \\
\mathbf{n}'_2 &= (\mathbf{V}_Q - \mathbf{V}_P) \times (\mathbf{V}_R - \mathbf{V}_S)
\end{aligned}$$

The unit direction of $\mathbf{w}_c(t)$ can then be expressed as

$$\mathbf{n}_E(t) = \frac{\mathbf{n}'_0 + \mathbf{n}'_1 t + \mathbf{n}'_2 t^2}{|\mathbf{n}'_0 + \mathbf{n}'_1 t + \mathbf{n}'_2 t^2|} \tag{B.4}$$

We can now find the impulse during the time interval h is [73],

$$\mathbf{I} = k_s \int_t^{t+h} \mathbf{n}_E(t)^T \left[\Delta d\, \mathbf{n}_E(t) - \mathbf{w}_c(t)\right] \mathbf{n}_E(t) dt \tag{B.5}$$

Since the time step h is very small in our model, we can get the approximations for $\mathbf{n}_E(t)$ and $\mathbf{w}_c(t)$ as [73],

$$\begin{aligned}
\mathbf{w}_c(t) &= \omega_P(t)\mathbf{P}(t) + \omega_Q(t)\mathbf{Q}(t) - \omega_R(t)\mathbf{R}(t) - \omega_S(t)\mathbf{S}(t) \\
&\approx \mathbf{w}_c^{app}(t) = \omega_{P_0}\mathbf{P}(t) + \omega_{Q_0}\mathbf{Q}(t) - \omega_{R_0}\mathbf{R}(t) - \omega_{S_0}\mathbf{S}(t)
\end{aligned}$$

where $\omega_{P_0} = 1 - \dfrac{b_0e_0 - c_0d_0}{a_0c_0 - b_0^2}$, $\omega_{Q_0} = \dfrac{b_0e_0 - c_0d_0}{a_0c_0 - b_0^2}$, $\omega_{R_0} = 1 - \dfrac{a_0e_0 - b_0d_0}{a_0c_0 - b_0^2}$, and $\omega_{S_0} = \dfrac{a_0e_0 - b_0d_0}{a_0c_0 - b_0^2}$, and

$$\begin{aligned}\mathbf{n}_E(t) = \frac{\mathbf{n}'_0 + \mathbf{n}'_1 t + \mathbf{n}'_2 t^2}{|\mathbf{n}'_0 + \mathbf{n}'_1 t + \mathbf{n}'_2 t^2|} &\approx \mathbf{n}_E^{app}(t) = \frac{\mathbf{n}'_0 + \mathbf{n}'_1 t + \mathbf{n}'_2 t^2}{|\mathbf{n}'_0|} \\ &= \frac{\mathbf{n}'_0}{|\mathbf{n}'_0|} + \frac{\mathbf{n}'_1}{|\mathbf{n}'_0|}t + \frac{\mathbf{n}'_2}{|\mathbf{n}'_0|}t^2 = \mathbf{n}_0 + \mathbf{n}_1 t + \mathbf{n}_2 t^2\end{aligned}$$

where $\mathbf{n}_0 = \dfrac{\mathbf{n}'_0}{|\mathbf{n}'_0|}$, $\mathbf{n}_1 = \dfrac{\mathbf{n}'_1}{|\mathbf{n}'_0|}$, and $\mathbf{n}_2 = \dfrac{\mathbf{n}'_2}{|\mathbf{n}'_0|}$. And thus we can get the discrete impulse during the time interval $[t_1, t_2]$ as

$$\begin{aligned}\mathbf{I}_d^{dis} = \frac{h}{2}\Big\{&\left(\mathbf{n}_E^{app}(t_1)\right)^T \left[\Delta d\, \mathbf{n}_E^{app}(t_1) - \mathbf{w}_c^{app}(t_1)\right] \mathbf{n}_E^{app}(t_1) \\ &+ \left(\mathbf{n}_E^{app}(t_2)\right)^T \left[\Delta d\, \mathbf{n}_E^{app}(t_2) - \mathbf{w}_c^{app}(t_2)\right] \mathbf{n}_E^{app}(t_2)\Big\}\end{aligned} \tag{B.6}$$

So the continuous penalty forces created during the time interval $[t_1, t_2]$ in numerical scheme can be expressed as

$$\begin{aligned}\mathbf{F}_c^{dis} = \frac{\mathbf{I}_d^{dis}}{h} = \frac{1}{2}\Big\{&\left(\mathbf{n}_E^{app}(t_1)\right)^T \left[\Delta d\, \mathbf{n}_E^{app}(t_1) - \mathbf{w}_c^{app}(t_1)\right] \mathbf{n}_E^{app}(t_1) \\ &+ \left(\mathbf{n}_E^{app}(t_2)\right)^T \left[\Delta d\, \mathbf{n}_E^{app}(t_2) - \mathbf{w}_c^{app}(t_2)\right] \mathbf{n}_E^{app}(t_2)\Big\}\end{aligned} \tag{B.7}$$

APPENDIX C

MATLAB CODE FOR THE BEAM BUCKLING PROBLEM WITH DIVERGENT CHANNEL CONSTRAINTS

In this part, we give an introduction to the code for the static simulation of a buckled Euler beam with divergent channel constraints. The following table C.1 shows the key variables in the Matlab code. Table C.2 shows the structure of the code.

```
clear all;
clc;

R=1;
L=R*pi/2;

EI=1;
% section
NS=8;
wds=zeros(NS,1);
% domain
NL=50;
ds=L/NL;
% range NK must be odd number
NK=41;
STARTI=21;

Tx=-500.0;
Ty=-500.0;

```

R	Radius of channel centerline
L	Length of the beam
EI	Bending stiffness
NS	The constrain is discretized into NS straight channels
NL	The beam is discretized into NL discrete elements
NK	The range is discretized into NK possible values
ds	The length of each discrete beam element
Tx	The load applied at the end of beam in x direction
Ty	The load applied at the end of beam in y direction
sur_x	The survival storing the x position of the beam configuration in local coordinate system of each straight channel constraints
sur_y	The survival storing the y position of the beam configuration in local coordinate system of each straight channel constraints
$sur\_section$	The survival storing the *section* location of the current beam node
sur_x_old	The x survival in previous stage
sur_y_old	The y survival in previous stage
$sur\_section\_old$	The *section* survival in previous stage
$cost$	The total potential energy of the current survival
$cost\_old$	The total potential energy of the previous survival
$configx$	The x position of beam configuration with minimum potential energy in global coordinate system
$configy$	The y position of beam configuration with minimum potential energy in global coordinate system

Table C.1: The key variables in the Matlab code for the beam buckling problem with divergent channel constraints based on Viterbi algorithm.

```
1.  Initialization (line 1 ~ 19): assign the geometric and material properties,
    and set the discretization strategies for domain, range, and constraints
2.  Constraints discretization (line 24 ~ 127): discretize the continuous con-
    straint into $NS$ straight channels
3.  Range discretization (line 166 ~ 170)
3.  Calculate the potential energy of survival in the first two stages line 184 ~
    208
4.  Calculate the potential energy of survival in the first two stages line 210 ~
    292
5.  for $stgi = 4 \sim NL$ (stage $4 \sim NL$: line 294 ~ 453)
        Calculate the cost of all the possible survivals in current stage
        line 323 ~ 410:
        for $ppsi = 1 \sim NK$ (range number in pre-pre-stage)
            for $psi = 1 \sim NK$ (range number in pre-stage)
                for $si = 1 \sim NK$ (range number in current stage)
                  Calculate $cost\_temp(ppsi, psi, si)$ based on equation (3.14)
                end
            end
        end
        Find survivals with minimum cost among $cost\_temp(ppsi, psi, si)$
        for $psi = 1 \sim NK$ (line 412 ~ 451)
            for $si = 1 \sim NK$
                Calculate the cost of the $survival(psi, si)$ in current stage:
                $cost(psi, si) = \min cost\_temp(1 \sim NK, psi, si)$
                Record the survivals path: $sur\_x$, $sur\_y$, $sur\_section$
            end
        end
    end
6.  Find the survival with minimum potential energy
    $min\_cost = cost(min\_psi, min\_si) = \min cost(1 \sim NK, 1 \sim NK)$
    $final\_survival = [survival\_x, y, section(min\_psi, min\_si)]$
7.  Transfer the local position information into global coordinate system
    $configx, configy = rotation\,(final\_survival)$
```

Table C.2: The algorithm structure of the Matlab code for the beam buckling problem with divergent channel constraints.

```
feature('accel','on')
tic
%% %%%% initial constraints (constraints discretization) %%%%%%
DeltaRs=0.05;
DeltaRe=0.10;
DLR=0.05;

theta=linspace(0,pi/2);

% lower wall
low_x=(R+DLR)*sin(theta);
low_y=R-(R+DLR)*cos(theta);
deltaup=DeltaRe-DeltaRs;
drtheta=deltaup*theta/(pi/2);
% section lower wall
slower_x=(R-DeltaRs-drtheta).*sin(theta);
slower_y=R-(R-DeltaRs-drtheta).*cos(theta);
R_cx=R*sin(theta);
R_cy=R-R*cos(theta);
dtheta=pi/2/(NS-1);
Stheta=zeros(NS,1);
Stheta(1)=0.0;
for i=2:NS
Stheta(i)=Stheta(i-1)+dtheta;
end

% discrete lower wall
seclx=zeros(NS+1,1);
secly=zeros(NS+1,1);
seclx(1)=0.0;
```

```
secly(1)=-DLR;
seclx(2)=seclx(1)+(R+DLR)*tan(dtheta/2);
secly(2)=secly(1);
for i=3:NS
seclx(i)=seclx(i-1)+2*(R+DLR)*tan(dtheta/2)*cos(Stheta(i-1));
secly(i)=secly(i-1)+2*(R+DLR)*tan(dtheta/2)*sin(Stheta(i-1));
end
seclx(NS+1)=seclx(NS)+(R+DLR)*tan(dtheta/2)*cos(Stheta(NS));
secly(NS+1)=secly(NS)+(R+DLR)*tan(dtheta/2)*sin(Stheta(NS));

% discrete upper wall

% the calibration point-mid point of each section:2~NS-1; 1st and
    last
% section, the calibration points are the start and end point
    respectively
xcal=zeros(NS,1);
ycal=zeros(NS,1);
xcal(1)=0.0;
ycal(1)=DeltaRs;
wds(1)=DLR+DeltaRs;
for i=2:NS-1
thetai=dtheta*(i-1);
xcal(i)=(R-DeltaRs-deltaup*(thetai/(pi/2)))*sin(thetai);
ycal(i)=R-(R-DeltaRs-deltaup*(thetai/(pi/2)))*cos(thetai);
wds(i)=DLR+DeltaRs+deltaup*(thetai/(pi/2));
end
xcal(NS)=R-DeltaRs-deltaup;
ycal(NS)=R;
wds(NS)=DLR+DeltaRs+deltaup;
```

```
% section upper
secsup_x=zeros(NS+1,1);
secsup_y=zeros(NS+1,1);
secsup_x(1)=0.0;
secsup_y(1)=DeltaRs;
for i=2:NS-1
secsup_x(i)=(ycal(i-1)+xcal(i)*tan(Stheta(i))-xcal(i-1)*...
tan(Stheta(i-1))-ycal(i))/(tan(Stheta(i))-tan(Stheta(i-1)));
secsup_y(i)=ycal(i)+(secsup_x(i)-xcal(i))*tan(Stheta(i));
end
secsup_x(NS)=R-DeltaRs-deltaup;
secsup_y(NS)=ycal(NS-1)+(secsup_x(NS)-xcal(NS-1))*tan(Stheta(NS-1));
secsup_x(NS+1)=xcal(NS);
secsup_y(NS+1)=ycal(NS);

% relative section position
secx=zeros(4,NS);
secy=zeros(4,NS);
secx(:,1)=[seclx(1),seclx(2),secsup_x(2),secsup_x(1)];
secy(:,1)=[secly(1)+DLR,secly(2)+DLR,secsup_y(2)+DLR,secsup_y(1)+DLR
    ];
for i=2:NS-1
d2x=seclx(i+1)-seclx(i);
d2y=secly(i+1)-secly(i);
d3x=secsup_x(i+1)-seclx(i);
d3y=secsup_y(i+1)-secly(i);
d4x=secsup_x(i)-seclx(i);
d4y=secsup_y(i)-secly(i);
secx(:,i)=[0,d2x*cos(Stheta(i))...
+d2y*sin(Stheta(i)),d3x*cos(Stheta(i))...
+d3y*sin(Stheta(i)),d4x*cos(Stheta(i))+d4y*sin(Stheta(i))];
```

```
secy(:,i)=[0,-d2x*sin(Stheta(i))+d2y*cos(Stheta(i)),...
-d3x*sin(Stheta(i))+d3y*cos(Stheta(i)),...
-d4x*sin(Stheta(i))+d4y*cos(Stheta(i))];
end
d2x=seclx(NS+1)-seclx(NS);
d2y=secly(NS+1)+L/4-secly(NS);
d3x=secsup_x(NS+1)-seclx(NS);
d3y=secsup_y(NS+1)+L/4-secly(NS);
d4x=secsup_x(NS)-seclx(NS);
d4y=secsup_y(NS)-secly(NS);
secx(:,NS)=[0,d2x*cos(Stheta(NS))...
+d2y*sin(Stheta(NS)),d3x*cos(Stheta(NS))...
+d3y*sin(Stheta(NS)),d4x*cos(Stheta(NS))+d4y*sin(Stheta(NS))];
secy(:,NS)=[0,-d2x*sin(Stheta(NS))...
+d2y*cos(Stheta(NS)),-d3x*sin(Stheta(NS))...
+d3y*cos(Stheta(NS)),-d4x*sin(Stheta(NS))+d4y*cos(Stheta(NS))];
%% %%%%%%%%%%%%% Viterbi start  %%%%%%%%%%%
sur_x=zeros(NK,NK,NL);
sur_y=zeros(NK,NK,NL);
sur_section=zeros(NK,NK,NL);

sur_x_old=zeros(NK,NK,NL);
sur_y_old=zeros(NK,NK,NL);
sur_section_old=zeros(NK,NK,NL);

% link
lsx=zeros(NK,NK);
lsy=zeros(NK,NK);
lssec=zeros(NK,NK);

lex=zeros(NK,NK);
```

```
ley=zeros(NK,NK);
lesec=zeros(NK,NK);

langle=inf(NK,NK);

lsx_old=zeros(NK,NK);
lsy_old=zeros(NK,NK);
lssec_old=zeros(NK,NK);

lex_old=zeros(NK,NK);
ley_old=zeros(NK,NK);
lesec_old=zeros(NK,NK);
langle_old=inf(NK,NK);

% Potential Engergy
cost=inf(NK,NK);
cost_old=inf(NK,NK);

dr=wds/(NK-1);

for i=1:NS
nreach(i)=floor(ds/dr(i));
end

vrange=zeros(NK,NS);

for i=1:NS
vrange(:,i)=linspace(0,wds(i),NK);
end

```

```
rmin=STARTI-nreach(1);
if rmin<1
rmin=1;
end
rmax=STARTI+nreach(1);
if rmax>NK
rmax=NK;
end


% stage 1 and 2
for si=rmin:rmax
lsx(STARTI,si)=ds;
lsy(STARTI,si)=DLR;
lssec(STARTI,si)=1;

theta2=asin((vrange(si,1)-vrange(STARTI,1))/ds);
lex(STARTI,si)=ds+ds*cos(theta2);
%ley(STARTI,si)=DLR+ds*sin(theta2);
ley(STARTI,si)=vrange(si,1);
lesec(STARTI,si)=1;

langle(STARTI,si)=theta2;
theta2=theta2+Stheta(1);

% update cost
cost(STARTI,si)=PE(0,theta2,EI,NL,Tx,Ty);
end
sur_x(:,:,1)=lsx;
sur_y(:,:,1)=lsy;
```

```
sur_section(:,:,1)=lesec;

sur_x(:,:,2)=lex;
sur_y(:,:,2)=ley;
sur_section(:,:,2)=lesec;

% stage 3
% store previous stage information
cost_old=cost;
lsx_old=lsx;
lsy_old=lsy;
lssec_old=lssec;
lex_old=lex;
ley_old=ley;
lesec_old=lesec;
langle_old=langle;

sur_x_old=sur_x;
sur_y_old=sur_y;
sur_section_old=sur_section;
% update current stage
for psi=1:NK
if cost_old(STARTI,psi)==inf
cost(psi,:)=inf(NK,1);
else
rmin=psi-nreach(lesec(STARTI,psi));
if rmin<1
rmin=1;
end
rmax=psi+nreach(lesec(STARTI,psi));
if rmax>NK
```

```
rmax=NK;
end

for si=rmin:rmax
lsx(psi,si)=lex_old(STARTI,psi);
lsy(psi,si)=ley_old(STARTI,psi);
lssec(psi,si)=lssec_old(STARTI,psi);

theta2=asin((vrange(si,1)-vrange(psi,1))/ds);
lex(psi,si)=lsx(psi,si)+ds*cos(theta2);
ley(psi,si)=lsy(psi,si)+ds*sin(theta2);
inside=inpolygon(lex(psi,si),ley(psi,si),secx(:,1),secy(:,1));
% end point in current section
if inside==1
lesec(psi,si)=lssec_old(STARTI,psi);
% end point in next section
else
% transform to the local coordinate system in next section
dsx=lsx(psi,si)-secx(2,lssec(psi,si));
dsy=lsy(psi,si)-secy(2,lssec(psi,si));
dtheta=Stheta(lssec(psi,si)+1)-Stheta(lssec(psi,si));
lxnew=dsx*cos(dtheta)+dsy*sin(dtheta);
lynew=-dsx*sin(dtheta)+dsy*cos(dtheta);

% update the end position of link(psi,si)
seci=lssec(psi,si)+1;
if abs(vrange(si,seci)-lynew) > ds
test=inf;
continue;
end
theta2=asin((vrange(si,seci)-lynew)/ds);
```

```

% relative position
lex(psi,si)=lxnew+ds*cos(theta2);
ley(psi,si)=lynew+ds*sin(theta2);
% location of section
lesec(psi,si)=lssec(psi,si)+1;
end

langle(psi,si)=theta2;

% update cost
theta2=theta2+Stheta(lesec(psi,si));
theta1=langle_old(STARTI,psi)+Stheta(lesec_old(STARTI,psi));
cost(psi,si)=cost_old(STARTI,psi)+PE(theta1,theta2,EI,NL,Tx,Ty);

% update survivor
sur_x(psi,si,1:2)=sur_x_old(STARTI,psi,1:2);
sur_y(psi,si,1:2)=sur_y_old(STARTI,psi,1:2);
sur_section(psi,si,1:2)=sur_section(STARTI,psi,1:2);

sur_x(psi,si,3)=lex(psi,si);
sur_y(psi,si,3)=ley(psi,si);
sur_section(psi,si,3)=lesec(psi,si);
end
end
end

% stage 4 ~ NL
stagei=3;
for stgi=4:NL
```

```
stagei=stgi;
disp(stagei);
% store previous stage information
cost_old=cost;
lsx_old=lsx;
lsy_old=lsy;
lssec_old=lssec;
lex_old=lex;
ley_old=ley;
lesec_old=lesec;
langle_old=langle;

sur_x_old=sur_x;
sur_y_old=sur_y;
sur_section_old=sur_section;

cost_temp=inf(NK,NK,NK);
lsx_temp=zeros(NK,NK,NK);
lsy_temp=zeros(NK,NK,NK);
lssec_temp=zeros(NK,NK,NK);

lex_temp=zeros(NK,NK,NK);
ley_temp=zeros(NK,NK,NK);
lesec_temp=zeros(NK,NK,NK);

langle_temp=zeros(NK,NK,NK);
for ppsi=1:NK
for psi=1:NK
if cost_old(ppsi,psi)==inf
%                    cost_temp(ppsi,psi,:)=inf(1,NK);
continue;
```

```
end

rmin=psi-nreach(lesec(ppsi,psi));
if rmin<1
rmin=1;
end
rmax=psi+nreach(lesec(ppsi,psi));
if rmax>NK
rmax=NK;
end
for si=rmin:rmax
lsx_temp(ppsi,psi,si)=lex_old(ppsi,psi);
lsy_temp(ppsi,psi,si)=ley_old(ppsi,psi);
lssec_temp(ppsi,psi,si)=lesec_old(ppsi,psi);

seci=lesec_old(ppsi,psi);
theta2=asin((vrange(si,seci)-lsy_temp(ppsi,psi,si))/ds);

lex_temp(ppsi,psi,si)=lsx_temp(ppsi,psi,si)+ds*cos(theta2);
ley_temp(ppsi,psi,si)=lsy_temp(ppsi,psi,si)+ds*sin(theta2);
inside=inpolygon(lex_temp(ppsi,psi,si),...
ley_temp(ppsi,psi,si),secx(:,seci),secy(:,seci));
% end point in current section
if inside==1
lesec_temp(ppsi,psi,si)=seci;
% angle in global coordinates system
theta2g=theta2+Stheta(seci);
% end point in next section
else
% forward or backward?
gendx=seclx(seci)...
```

```
+lex_temp(ppsi,psi,si)*cos(Stheta(seci))...
-ley_temp(ppsi,psi,si)*sin(Stheta(seci));
gendy=secly(seci)...
+lex_temp(ppsi,psi,si)*sin(Stheta(seci))...
+ley_temp(ppsi,psi,si)*cos(Stheta(seci));

thetaend=pi/2-atan((R-gendy)/gendx);
thetasec=pi/2-atan((R-ycal(seci))/xcal(seci));

if thetaend>thetasec
seci_new=seci+1;
else
seci_new=seci-1;
end
% transform to the local coordinate system in next section
gsx=lsx_temp(ppsi,psi,si)*cos(Stheta(seci))...
-lsy_temp(ppsi,psi,si)*sin(Stheta(seci));
gsy=lsx_temp(ppsi,psi,si)*sin(Stheta(seci))...
+lsy_temp(ppsi,psi,si)*cos(Stheta(seci));

dsx=-seclx(seci_new)+seclx(seci)+gsx;
dsy=-secly(seci_new)+secly(seci)+gsy;
lxnew= dsx*cos(Stheta(seci_new))+dsy*sin(Stheta(seci_new));
lynew=-dsx*sin(Stheta(seci_new))+dsy*cos(Stheta(seci_new));



if abs(vrange(si,seci_new)-lynew) > ds
cost_temp(ppsi,psi,si)=inf;
continue;
end
theta2=asin((vrange(si,seci_new)-lynew)/ds);
```

```

% relative position
lex_temp(ppsi,psi,si)=lxnew+ds*cos(theta2);
ley_temp(ppsi,psi,si)=lynew+ds*sin(theta2);
% location of section
lesec_temp(ppsi,psi,si)=seci_new;

% angle in global coordinates system
theta2g=theta2+Stheta(seci_new);
end

langle_temp(ppsi,psi,si)=theta2;

% update cost
theta1=langle_old(ppsi,psi)+Stheta(lesec_old(ppsi,psi));
cost_temp(ppsi,psi,si)=cost_old(ppsi,psi)...
+PE(theta1,theta2g,EI,NL,Tx,Ty);
end
end
end

for psi=1:NK
for si=1:NK
temp_min_cost=inf;
temp_min_ppsi=inf;
for ppsi=1:NK
if temp_min_cost>cost_temp(ppsi,psi,si)
temp_min_cost=cost_temp(ppsi,psi,si);
temp_min_ppsi=ppsi;
end
```

```
end

 if temp_min_ppsi<inf
 cost(psi,si)=temp_min_cost;
 lsx(psi,si)=lex_old(temp_min_ppsi,psi);
 lsy(psi,si)=ley_old(temp_min_ppsi,psi);
 lssec(psi,si)=lesec_old(temp_min_ppsi,psi);

 lex(psi,si)=lex_temp(temp_min_ppsi,psi,si);
 ley(psi,si)=ley_temp(temp_min_ppsi,psi,si);
 lesec(psi,si)=lesec_temp(temp_min_ppsi,psi,si);

 langle(psi,si)=langle_temp(temp_min_ppsi,psi,si);

% update survivor
sur_x(psi,si,1:stgi-2)=sur_x_old(temp_min_ppsi,psi,1:stgi-2);
sur_y(psi,si,1:stgi-2)=sur_y_old(temp_min_ppsi,psi,1:stgi-2);
sur_section(psi,si,1:stgi-2)=sur_section_old(temp_min_ppsi,psi,1:stgi
    -2);

sur_x(psi,si,stgi-1)=lsx(psi,si);
sur_y(psi,si,stgi-1)=lsy(psi,si);
sur_section(psi,si,stgi-1)=lssec(psi,si);

sur_x(psi,si,stgi)=lex(psi,si);
sur_y(psi,si,stgi)=ley(psi,si);
sur_section(psi,si,stgi)=lesec(psi,si);
 else
 cost(psi,si)=inf;
end
end
```

```
end

end


mincost=inf;
for psi=1:NK
for si=1:NK
if mincost>cost(psi,si)
mincost=cost(psi,si);
min_psi=psi;
min_si=si;
end
end
end

configx=zeros(1,NL);
configy=zeros(1,NL);

mincost
for i=1:NL
if mincost==inf
min_psi=1;
min_si=1;
seci=1;
else
seci=sur_section(min_psi,min_si,i);
theta=Stheta(seci);
configx(i)=seclx(seci)+sur_x(min_psi,min_si,i)*cos(Stheta(seci))...
-sur_y(min_psi,min_si,i)*sin(Stheta(seci));
configy(i)=secly(seci)+sur_x(min_psi,min_si,i)*sin(Stheta(seci))...
```

```
+sur_y(min_psi,min_si,i)*cos(Stheta(seci));
end
end

toc


%% %%%%%%%%%%%% Viterbi end  %%%%%%%%%%%%%%

plot(low_x,low_y,'-c',R_cx,R_cy,'-.k',slower_x,slower_y,'-c',seclx,
    secly,'.-b',secsup_x,secsup_y,'.-b',[0.0,configx],[0.0,configy],'
    .-r');
grid on;
axis equal;
xlim([0  1.2]);
ylim([-0.2  1.2]);
```

codes/Matlab_code_for_the_beam.m

```
function  Potential=PE(theta1,theta2,EI,NL,Tx,Ty)
Potential=(EI*0.5*(theta2-theta1)^2*(NL+2)*(NL+2)-Ty*sin(theta2)+Tx*
    (1-cos(theta2)))/(NL+2);
end
```

codes/cost_function.m

APPENDIX D

C++ CODE FOR THE REAL TIME SIMULATION OF KIRCHHOFF ROD

In this part, we give an introduction to the code of real time simulation of surgical thread (the application of the model based on the theories of Kirchhoff rod). The key variables are shown in the following:

M	Total mass of the surgical thread. (Unit: Kg)
L	Length of the surgical thread. (Unit: m)
N	The surgical thread is discretized into N segments
m	The mass of each discrete node of the surgical thread. (Unit: Kg)
l	The length of each discrete segments of the surgical thread. (Unit: m)
$BendStif$	Bending stiffness (Unit: $N \cdot m^2$)
$TorsionStiff$	Torsion stiffness (Unit: $N \cdot m^2$)
Q	The total torsion angle of the thread
h	Time step. (Unit: $second$)
$RunTime$	The total simulation time
$MatrixXd$ X	The position matrix $(3 \times N)$ of discrete surgical thread's nodes . (Unit: m)
$MatrixXd$ DX	The discrete segment displacements matrix $(3 \times N-1)$: $DX(:,i) = X(:,i+1) - X(:,i)$, $i = 1 \ldots N-1$. (Unit: m)
$MatrixXd$ EX	The unit direction matrix $(3 \times N - 1)$ of discrete surgical thread: $EX(:,i) = DX(:,i)/\lvert DX(:,i)\rvert$, $i = 1 \ldots N-1$
$MatrixXd$ V	The speed matrix $(3 \times N)$ of discrete surgical thread's nodes. (Unit: m/s)
$DispAir$	The air dissipation coefficient k_{air} in equation (4.24)
$DispBend$	The bending dissipation coefficient k_d in equation (4.25)

MatrixXd Fext	The forces matrix ($3 \times N$) applied on the discrete nodes. (Unit: N)
MatrixXd FGrav	The gravity matrix ($3 \times N$) of the discrete nodes. (Unit: N)
MaxCols	The number of maximum cells can be taken by the discrete surgical thread
DeltDis	The criteria distance of collision detection Δd
DeltaCellGridSize	The cell size l_{cell} in the uniform space partitioning
SpringConst	The spring constant k_p in equation (4.53) for the penalty force (Unit: N/m)
MaxIter	The number of maximum iterations can been taken before the Picard iteration gets convergent

The algorithm structure of the code is shown in the following:

```
1.  initialization(); Initialize the simulation. (line 371 in the code file: main.cpp)
    while (run)
2.      UpdateExtForce(); Calculate the forces applied on each nodes (line 190 in the code file: main.cpp)
3.      CollisionDetection(); (line 190 in the code file: main.cpp)
4.      CollisionResponse(); (line 199 in the code file: main.cpp)
5.      UpdatePosition(); Employ Picard iteration to update the positions at current time step (line 208 in the code file: main.cpp)
    end
```

The function of initialization() is very straight forward; also UpdateExtForce(), CollisionResponse(), and UpdatePosition() are directly following the work in chapter 3, so we are also not going to further explain them. Here we discuss CollisionDetection() in detail. The key variables and the algorithm structure of the collision detection are shown in the following:

$NodsCellLocs$	The cell location of each discrete node
$NodsCellErrs$	The position of each discrete node relative to the coordinate system assigned to the current cell occupied
$ColCheckFlg$	The self collision check status between two segments of the surgical thread. If checked, $ColCheckFlg = 1$, else $ColCheckFlg = 0$
$SelfColSta$	The self collision status between two segments of the surgical thread. If collide, $SelfColSta = 1$, else $SelfColSta = 0$
$CellNeighbors$	The neighbor cells positions relative to current one.
$CellBeTakenNum$	The total number of the cells that are occupied by the discrete surgical thread
$CellBeTaken$	The positions of the cells that are occupied by the discrete surgical thread
$CellTakenSparse$	A sparse matrix that served as index from the cell position to $SegsInCell$
$SegsInCell$	Recording the segments information in each cell has been occupied
$SegsInCellNum$	The number of total segments in each cell has been occupied

```
1.  Do space partitioning based on the algorithm shown in Table 4.1. (line
    41 ~ 303 in the code file: CollisionDetection.h)
    for $i = 0 \cdots CellBeTakenNum$
2.      Check the self collisions in current cell
        (line 313 ~ 393 in CollisionDetection.h)
        for $j = 0 : SegsInCellNum(i)$
            for $k = j + 1 : SegsInCellNum(i)$
                if the distance between to segments smaller than $\Delta d$
                    $SelfColSta = 1$ Collision happens
                end
                $ColCheckFlg = 1$ Collision has been checked
            end
        end
3.      Check self collisions in its neighbor cells ($CellNeighbors$)
        (line 395 ~ 497 in CollisionDetection.h)
        for $j = 0 : SegsInCellNum(i)$
            if the distance between to segments smaller than $\Delta d$
                $SelfColSta = 1$ Collision happens
            end
            $ColCheckFlg = 1$ Collision has been checked
        end
    end
```

```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <glut.h>

#include <Eigen/Eigen>
#include <Eigen/SparseLU>
#include <Eigen/Sparse>
#include <ctime>
//#include <tbb/tbb.h>

 using namespace std;
 using namespace Eigen;


#define M 0.01    //total mass
#define L 1.0  // length of the thread
#define h 0.001  // time step
#define g 9.81
#define DIM 3
#define Error 1e-6   // numerical error caused by float calculation
#define MaxIter 1000
// 4*(N-1): max cells can be taken; 3*MaxCols: the cells position
#define MaxCols 10
#define PI 3.1415926

#define runningtimecheck
//#define drawcellbetaken
#define drawgrid
#define SparseMatrixSolver
#define SparseVectorCellIndex
```

```
#define SparseColDetect
//#define inextcheck
//#define outputtorque

#define N 101      // the number of discrete nodes of the thread
#define BendStif 0.00025  //#define BendStif 0.02             // 1->
    0.001: 2-3 plectonemes;  2-> 0.0005: 1 plectoneme; 3 -> 0.00025
    no plectorneme
#define TorsionStiff 0.001 //#define TorsionStiff 0.02
#define SpringConst 4 //#define SpringConst 200
double RunTime=40;

// special load
double Q=0.0;             // initial twisting angle
double QEnd=50*PI;
double Qv=0.2*(N-1);        // the velocity of twisting angle

//test
double Q=0.0;             // initial twisting angle
double QEnd=250*PI;
double Qv=0.2*(N-1);        // the velocity of twisting angle

#if 20*l>1
double XSTOP=0.8*l;  // stop moving the end of thread (special load
    end)
double DeltDis=0.2*l; // collision detection criterion distance
#define MaxCellPerSeg 6
#else
double XSTOP=0.5;
double DeltDis=0.01;
#define MaxCellPerSeg 6
```

```
#endif
double DeltaCellGridSize=DeltDis; //(if change this DeltaCellGridSize
    , also change the dimension of SegsCellLocation = ceil(1/0.3)=4)


#ifdef SparseMatrixSolver
//#define ParallelComputing
typedef Eigen::SparseMatrix<double> SpMat; // declares a column-major
     sparse matrix type of double
typedef Eigen::Triplet<double> T;
#endif

#ifdef SparseVectorCellIndex
typedef Eigen::SparseVector<int> SpVecInt;
#endif

double m=double(M/N);          // mass of each node
double l=double(L/(N-1));     // length of each segment
double lsqu=l*l;
double lsquRec=1/(l*l);
double lunit=1/l;
double hRec=1/h;
double hsqu=h*h;
double DispAir=0.2/(N-1);   //air dissipation
double DispBend=0.01*m;        //bending dissipation


int QstoredSize=int(ceil(RunTime/h))+1; // store the total twisting
    angle of the thread
// special load start

```

```
Vector3d VeEnd=Vector3d::Zero();       // Input: the velocity of the
    thread end
Vector3d XEnd=Vector3d::Zero();      //  the position of the end of
    the thread


MatrixXd X=MatrixXd::Zero(DIM,N);  //position of each discrete node
MatrixXd DX=MatrixXd::Zero(DIM,N-1);  //the vector of the discrete
    segments
MatrixXd EX=MatrixXd::Zero(DIM,N-1);  //the unit direction of the
    discrete segments

MatrixXd V=MatrixXd::Zero(DIM,N);  //velocity of each discrete node
//MatrixXd VSqureNorm=MatrixXd::Zero(DIM,N); // square norm of the
    velocity of each node

VectorXd Qstore=VectorXd::Zero(QstoredSize);

MatrixXd Fext=MatrixXd::Zero(DIM,N);   // external forces applied on
    each node
MatrixXd FGrav=MatrixXd::Zero(DIM,N);   // gravaity forces

Vector3d UnitVectorx(1,0,0);              // unit vector

// variables in the filečž CollisionDetection  start
#define XWidthNeg MaxCellPerSeg*(N-1)
#define XWidthPos MaxCellPerSeg*(N-1)
#define YWidthNeg MaxCellPerSeg*(N-1)
#define YWidthPos MaxCellPerSeg*(N-1)
#define ZWidthNeg (N-1)
```

```
#define ZWidthPos (N-1)

int const XWid=XWidthNeg+XWidthPos;
int const YWid=YWidthNeg+YWidthPos;
int const ZWid=ZWidthNeg+ZWidthPos;
int const XYWid=XWid*YWid;
double const XMin=-XWidthNeg*DeltaCellGridSize;
double const YMin=-YWidthNeg*DeltaCellGridSize;
double const ZMin=-ZWidthNeg*DeltaCellGridSize;
#define incell(xx,yy,zz) xx+yy*XWid+zz*XYWid
#define zcell(xx) int(xx/XYWid)
#define ycell(xx) int((xx%XYWid)/XWid)
#define xcell(xx) (xx%XYWid)%XWid

int ColCheckFlgIni[(N-1)*(N-1)]={0};
// variables in the file: CollisionDetection end



#ifdef runningtimecheck
double cellingfloor=0.0;
double coldecting=0.0;
double Bresenham3Dtime=0.0;
double picardtime=0.0;
double solvingeqtime=0.0;
double elapsed_secs=0.0;
double upextftime=0.0;
double coldettime=0.0;
double colrestime=0.0;
double uppostime=0.0;
#endif

```

```
#ifdef drawcellbetaken
//draw
int drawcellnumber=0;
int drawcell[MaxCellPerSeg*(N-1)]={0};
#endif

#ifdef outputtorque
ofstream myfile;
#endif

double TimeI=0.0;
int TimeIter=0;

#include "initialization.h"
#include "UpdateExtForce.h"
#include "CollisionDetection.h"
#include "CollisionResponse.h"
#include "UpdatePosition.h"

clock_t start;
void thread()
{

if (TimeI==0)
{
start= clock();
}

if (TimeIter%500==0)
{
cout<<TimeI<<" --> "<<Q<<"   ";
}

// 1
#ifdef runningtimecheck
clock_t begin = clock();
#endif
UpdateExtForce();

#ifdef runningtimecheck
clock_t end = clock();
upextftime =upextftime+ double(end - begin) / CLOCKS_PER_SEC;
begin = clock();
#endif

// 2
CollisionDetection();

#ifdef runningtimecheck
end = clock();
coldettime =coldettime+ double(end - begin) / CLOCKS_PER_SEC;
begin = clock();
#endif

// 3
CollisionResponse();

#ifdef runningtimecheck
end = clock();
colrestime =colrestime+ double(end - begin) / CLOCKS_PER_SEC;
begin = clock();
#endif
```

```
// 4
UpdatePosition();

#ifdef runningtimecheck
end = clock();
uppostime =uppostime+ double(end - begin) / CLOCKS_PER_SEC;
#endif
Qstore(TimeIter)=Q;

if (TimeI>=RunTime-h)
{
clock_t end = clock();
double elapsed_secs =double(end - start) / CLOCKS_PER_SEC;
cout<<endl<<"Running time = "<<elapsed_secs<<endl;
#ifdef runningtimecheck
  double totalsum=upextftime+coldettime+colrestime+uppostime;

  cout<<"UpdateExtForce()=  "<< upextftime<<" "<< upextftime/totalsum
    *100<<"%"<<endl;
  cout<<"CollisionDetection()=  "<< coldettime<<" "<< coldettime/
    totalsum*100<<"%"<<endl;
  cout<<"       Celling-> Cell Floor= "<<cellingfloor<<"
    Bresenham3Dtime=  "<<Bresenham3Dtime<<endl;
  cout<<"       Celling-> Detecting+= "<<coldecting <<endl;
  cout<<"CollisionResponse()= "<< colrestime<<" "<< colrestime/
    totalsum*100<<"%"<<endl;
  cout<<"UpdatePosition()=  "<< uppostime<<"  "<< uppostime/totalsum*
    100<<"%"<<endl;
  cout<<"       UpdatePosition->Pircardtime=  "<<picardtime<<"  sovling
     equation time=  "<<solvingeqtime<<endl;
```

```

   cout<<"Sum=        "<<totalsum<<endl;
#endif
std::system("pause");
exit(0);
}

#ifdef outputtorque

if (TimeIter%50==0)
{
myfile<<Q<<"   ";
}

if (TimeI>=RunTime-h){
myfile.close();
}
#endif

TimeIter+=1;
TimeI+=h;
}



void Draw() {
thread();
MatrixXd drawx=X;
drawx.row(0).array()=drawx.row(0).array()+XMin;
drawx.row(1).array()=drawx.row(1).array()+YMin;
drawx.row(2).array()=drawx.row(2).array()+ZMin;

```

```
glClear(GL_COLOR_BUFFER_BIT);
#ifdef drawcellbetaken
// draw cell been taken
glBegin(GL_QUADS);
glColor3f(0.0, 1.0, 1.0);
for (int i=0;i<drawcellnumber;i++)
{
   glVertex3f(xcell(drawcell[i])*DeltaCellGridSize+XMin,            ycell
     (drawcell[i])*DeltaCellGridSize+YMin,                    zcell(drawcell
     [i])*DeltaCellGridSize+ZMin);
   glVertex3f(xcell(drawcell[i])*DeltaCellGridSize+XMin+
     DeltaCellGridSize, ycell(drawcell[i])*DeltaCellGridSize+YMin,
                zcell(drawcell[i])*DeltaCellGridSize+ZMin);
   glVertex3f(xcell(drawcell[i])*DeltaCellGridSize+XMin+
     DeltaCellGridSize, ycell(drawcell[i])*DeltaCellGridSize+YMin+
     DeltaCellGridSize,      zcell(drawcell[i])*DeltaCellGridSize+ZMin);
   glVertex3f(xcell(drawcell[i])*DeltaCellGridSize+XMin,            ycell
     (drawcell[i])*DeltaCellGridSize+YMin+DeltaCellGridSize,      zcell(
     drawcell[i])*DeltaCellGridSize+ZMin);
}


glColor3f(0.0, 0.0, 1.0);
for (int i=0;i<N-1;i++)
{
   glVertex3f(NodsCellLocs[3*i]*DeltaCellGridSize+XMin,
     NodsCellLocs[3*i+1]*DeltaCellGridSize+YMin,
     NodsCellLocs[3*i+2]*DeltaCellGridSize+ZMin);
   glVertex3f(NodsCellLocs[3*i]*DeltaCellGridSize+XMin+
     DeltaCellGridSize,   NodsCellLocs[3*i+1]*DeltaCellGridSize+YMin,
                    NodsCellLocs[3*i+2]*DeltaCellGridSize+ZMin);
```

```
    glVertex3f(NodsCellLocs[3*i]*DeltaCellGridSize+XMin+
      DeltaCellGridSize,  NodsCellLocs[3*i+1]*DeltaCellGridSize+YMin+
      DeltaCellGridSize,   NodsCellLocs[3*i+2]*DeltaCellGridSize+ZMin);
    glVertex3f(NodsCellLocs[3*i]*DeltaCellGridSize+XMin,
      NodsCellLocs[3*i+1]*DeltaCellGridSize+YMin+DeltaCellGridSize,
      NodsCellLocs[3*i+2]*DeltaCellGridSize+ZMin);
 }
 glEnd();
#endif

#ifdef drawgrid
// draw cell grid
 glColor3f(1.0, 0.7, 0.8);
 glBegin(GL_LINES);
 double cell_x=-XWidthNeg*DeltaCellGridSize;
 for (int ii=0;ii<XWid;ii++)
 {
    glVertex3f(cell_x, -YWidthNeg*DeltaCellGridSize, 0.0);
    glVertex3f(cell_x, YWidthPos*DeltaCellGridSize, 0.0);
    cell_x+=DeltaCellGridSize;
 }

 double cell_y=-YWidthNeg*DeltaCellGridSize;
 for (int ii=0;ii<=YWid;ii++)
 {
    glVertex3f(-XWidthNeg*DeltaCellGridSize, cell_y, 0.0);
    glVertex3f(XWidthPos*DeltaCellGridSize, cell_y, 0.0);
    cell_y+=DeltaCellGridSize;
 }
 glEnd();

```

```
// Axis x-y-z
glColor3f(1.0, 0.0, 0.0);
glBegin(GL_LINES);
glVertex3f(-0.2, 0.0, 0.0);
glVertex3f(1.2, 0.0, 0.0);

glVertex3f(0.0, -0.6, 0.0);
glVertex3f(0.0, 0.6, 0.0);

glVertex3f(0.0, 0.0, -0.6);
glVertex3f(0.0, 0.0, 0.6);
glEnd();
#endif

// draw discrete thread
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_LINE_STRIP);
for (int ii=0;ii<=N-1;ii++)
{
  Vector3d temp=Vector3d(drawx.col(ii));
  glVertex3f(temp(0),temp(1),temp(2));
}
glEnd();

// draw discrete nodes of the thread
glColor3f(0.0, 0.0, 1.0);
glPointSize(5);
glBegin(GL_POINTS);
for (int ii=0;ii<=N-1;ii++)
{
  Vector3d temp=Vector3d(drawx.col(ii));
```

```
    glVertex3f(temp(0),temp(1),temp(2));
 }
 glEnd();

 glFlush();

 glutPostRedisplay();
 }

 void Initialize() {
 glClearColor(0.0, 0.0, 0.0, 0.0);
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glOrtho(-0.2, 1.2, -0.6, 0.2, -0.4, 0.4);
 //glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
 }

 int main(int iArgc, char** cppArgv) {

 //cout<<"size of ColCheckFlgIni="<<sizeof(ColCheckFlgIni)<<endl;

#ifdef outputtorque
 myfile.open ("torque.xls");

 myfile<<"Time"<<" ";
 for (double ii=0;ii<RunTime;ii=ii+50*h)
 {
 myfile<<ii<<" ";
 }
 myfile<<"\n"<<"Torque"<<" ";
#endif
```

```
// thread functions
initialization();

glutInit(&iArgc, cppArgv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(1000, 800);
glutInitWindowPosition(0, 0);
glutCreateWindow("XoaX.net");
Initialize();
glutDisplayFunc(Draw);

glutMainLoop();
return 0;
}
```

codes/main.cpp

```
//MatrixXd VSqureNorm(DIM,N);
//VectorXd VSqureNorm(N);
//MatrixXd BiNorm=MatrixXd::Zero(DIM,N-1); // this contains the cross
     product ei cross ei+1 vectors;
//VectorXd BetaCos=VectorXd::Zero(N-1); // this contains the dot
    product ei cross ei+1 vectors;
VectorXd BetaCosPre=VectorXd::Ones(N-1); // for our case its one
void UpdateExtForce()
{

  VectorXd VSqureNorm(N);
  VSqureNorm=V.colwise().squaredNorm();
  //---- update the current geometric information
  // update segment and its direction information
  DX=X.block(0,1,DIM,N-1)-X.block(0,0,DIM,N-1);
  EX=DX*lunit;

  // update dot product information
  VectorXd BetaCos(N-1);
  BetaCos(0)=1.00;
  BetaCos.segment(1,N-2)=(EX.block(0,0,DIM,N-2).array()*EX.block(0,1,
    DIM,N-2).array()).colwise().sum();
  //cout<<BetaCos<<endl;
#ifdef inextcheck
  //cheking the error caused by float calculation in CPU
  int temp_betag1=(BetaCos.array().abs() > 1).count();
  if ( temp_betag1>= 1)
  {
    //cout<<"temp_betag1="<<temp_betag1<<endl;
    int temp_betag2=(BetaCos.array().abs() > (1+Error*10000)).count()
   ;
    if ( temp_betag2>= 1)
    {
      std::cout<<BetaCos<<endl;
      std::cout<<"Error in angle calculation: "<<temp_betag2<<" angle
    dot products greater than 1"<<endl;
      std::system("pause");
      return 1;
    }
    else
    {
      //cout<<"before:"<<BetaCos.transpose()<<endl;
      BetaCos=BetaCos.unaryExpr(ptr_fun(GreaterThanOne));
      //cout<<"after:"<<BetaCos.transpose()<<endl;
      //system("pause");
    }
  }
#endif

  // update the binormal information
  MatrixXd BiNorm(DIM,N-1); // this contains the cross product ei
    cross ei+1 vectors;
  BiNorm.col(0)=UnitVectorx.cross(Vector3d(EX.col(0)));
  for (int ii=1;ii<=N-2;ii++)
  {
    BiNorm.col(ii)=Vector3d(EX.col(ii-1)).cross(Vector3d(EX.col(ii)))
    ;
  }


  //————————————————————— update external forces
    ————————————————————————————————————————————————//
  Fext.col(0)=-DispAir*VSqureNorm(0)*V.col(0)
            +BendStif*(-1/((1+BetaCos(1))*(1+BetaCos(1)))*DX.col(1))*
    lsquRec
            -DispBend*(-DX.col(1)*(BetaCos(1)-BetaCosPre(1)))*hRec*
    lsquRec;
  Fext.col(1)=-DispAir*VSqureNorm(1)*V.col(1)
            +BendStif*(1/((1+BetaCos(1))*(1+BetaCos(1)))*(DX.col(1)-DX.
    col(0))-1/((1+BetaCos(2))*(1+BetaCos(2)))*DX.col(2))*lsquRec
            -DispBend*((DX.col(1)-DX.col(0))*(BetaCos(1)-BetaCosPre(1))
    -DX.col(2)*(BetaCos(2)-BetaCosPre(2)))*hRec*lsquRec;
  for (int ii=2;ii<N-2;ii++)
  {
    Fext.col(ii)=-DispAir*VSqureNorm(ii)*V.col(ii)
            +BendStif*(1/((1+BetaCos(ii-1))*(1+BetaCos(ii-1)))*DX.col(
    ii-2)+1/((1+BetaCos(ii)   )*(1+BetaCos(ii)  ))*(DX.col(ii)-DX.col(
  ii-1))-1/((1+BetaCos(ii+1))*(1+BetaCos(ii+1)))*DX.col(ii+1))*
  lsquRec
              -DispBend*(DX.col(ii-2)*(BetaCos(ii-1)-BetaCosPre(ii-1))+(
  DX.col(ii)-DX.col(ii-1))*(BetaCos(ii)-BetaCosPre(ii))-DX.col(ii
  +1)*(BetaCos(ii+1)-BetaCosPre(ii+1)))*hRec*lsquRec;
}
Fext.col(N-2)=-DispAir*VSqureNorm(N-2)*V.col(N-2)
              +BendStif*(1/((1+BetaCos(N-4))*(1+BetaCos(N-3)))*DX.col(N
  -4)+1/((1+BetaCos(N-2))*(1+BetaCos(N-2)))*(DX.col(N-2)-DX.col(N
  -3)))*lsquRec
              -DispBend*(DX.col(N-4)*(BetaCos(N-3)-BetaCosPre(N-3))+(DX.
  col(N-2)-DX.col(N-3))*(BetaCos(N-2)-BetaCosPre(N-2)))*hRec*
  lsquRec;
Fext.col(N-1)=-DispAir*VSqureNorm(N-1)*V.col(N-1)
              +BendStif*(1/((1+BetaCos(N-2))*(1+BetaCos(N-2)))*DX.col(N
  -3))*lsquRec
              -DispBend*(DX.col(N-3)*(BetaCos(N-2)-BetaCosPre(N-2)))*hRec
  *lsquRec;

Fext=Fext+FGrav;

// update the external force --> add end load
// update end load information  --> only force no torque
XEnd=XEnd+h*VeEnd;
if(XEnd(0)<=XSTOP)
{
        XEnd(0)=XSTOP-Error;
        VeEnd=Vector3d::Zero();
}
// update the force
Fext.col(N-1)=Fext.col(N-1)+300*(XEnd-X.col(N-1));
```

```
// update the external force --> twisting
//Fext=MatrixXd::Zero(DIM,N);
// update end load information  --> only torque no force
if (VeEnd.norm()==0)
{
  Q=Q+Qv*h;
}
if (Q>QEnd)
{
  Qv=0.0;
}
// update the total torsional angle
for (int ii=1; ii<=N-2; ii++)
{
  Q=Q+Vector3d(BiNorm.col(ii)).dot(Vector3d(V.col(ii+1)-V.col(ii-1)
  ))*h/(1+BetaCos(ii));
}
Q=Q+Vector3d(BiNorm.col(0)).dot(Vector3d(V.col(1)))*h/(1+BetaCos(0)
  );
// calculate the force caused by torsion
double QPS=Q/(N-1);  // twist angle per segment
Fext.col(0)=Fext.col(0)-TorsionStiff*(-QPS*BiNorm.col(1)/(1+BetaCos
  (1)));
for (int ii=1; ii<=N-3; ii++)
{
   Fext.col(ii)=Fext.col(ii)-TorsionStiff*QPS*(BiNorm.col(ii-1)/(1+
  BetaCos(ii-1))-BiNorm.col(ii+1)/(1+BetaCos(ii+1)));
}
Fext.col(N-2)=Fext.col(N-2)-TorsionStiff*(QPS*BiNorm.col(N-3)/(1+
  BetaCos(N-3)));
  BetaCosPre=BetaCos;
  //std::cout<<"twisting: \n"<<Fext<<endl;
}
```

codes/UpdateExtForce.h

```
int NodsCellLocs[DIM*N]={0};
double NodsCellErrs[DIM*N]={0.0};
int ColCheckFlg[(N-1)*(N-1)]={0};
int SelfColSta[(N-1)*(N-1)]={0};
int CellNeighbours[13]={-1, -XWid+1, -XWid, -XWid-1,
        -XYWid+XWid+1,-XYWid+XWid,-XYWid+XWid-1,
        -XYWid+1,-XYWid, -XYWid-1,
        -XYWid-XWid+1,-XYWid-XWid,-XYWid-XWid-1};
int CellBeTakenNum=0;
int CellBeTaken[MaxCellPerSeg*(N-1)]={0};

#ifdef SparseVectorCellIndex
SpVecInt CellTakenSparse((XWidthNeg+XWidthPos)*(YWidthNeg+YWidthPos)*
    (ZWidthNeg+ZWidthPos));
#else
int CellTaken[(XWidthNeg+XWidthPos)*(YWidthNeg+YWidthPos)*(ZWidthNeg+
    ZWidthPos)]={0};
#endif
int CollisionDetection()
{
#ifdef runningtimecheck
clock_t begincoldet=clock();
#endif
#ifdef SparseVectorCellIndex
CellTakenSparse.setZero();
CellTakenSparse.reserve(MaxCellPerSeg*(N-1));
#else
for (int ii=0; ii<CellBeTakenNum; ii++)
{
CellTaken[CellBeTaken[ii]]=0;
}
#endif
//std::memset(&CellTaken[0],0,CellBeTakenNum);
CellBeTakenNum=0;

//fill(SegsInCellNum, SegsInCellNum+4*(N-1), 0);
int SegsInCellNum[MaxCellPerSeg*(N-1)]={0};
int SegsInCell[MaxCellPerSeg*(N-1)*MaxCols]={0};
copy(begin(ColCheckFlgIni),end(ColCheckFlgIni),begin(ColCheckFlg));
fill(SelfColSta, SelfColSta+(N-1)*(N-1), 0);


//--------------------    celling start -----------------------//

//Celling();

//MatrixXd temp_X=MatrixXd::Zero(3,N);


double DetlaDisRev=1/DeltaCellGridSize;
MatrixXd temp_loc=X*DetlaDisRev;
double temploca[DIM*N];
Eigen::Map<MatrixXd>(temploca,DIM,N) = temp_loc;
//temp_X=temp_X*DetlaDisRev;
//cout<<endl<<temp_X<<endl<<temp_loc;
//cout<<endl<<temp_loc<<endl;
```

```

for (int ii=0; ii<N; ii++)
{
NodsCellLocs[ii*DIM]=int(temploca[ii*DIM]);
NodsCellLocs[ii*DIM+1]=int(temploca[ii*DIM+1]);
NodsCellLocs[ii*DIM+2]=int(temploca[ii*DIM+2]);
NodsCellErrs[ii*DIM]=temploca[ii*DIM]-NodsCellLocs[ii*DIM];
NodsCellErrs[ii*DIM+1]=temploca[ii*DIM+1]-NodsCellLocs[ii*DIM+1];
NodsCellErrs[ii*DIM+2]=temploca[ii*DIM+2]-NodsCellLocs[ii*DIM+2];
}

#ifdef runningtimecheck
clock_t endcell1=clock();
cellingfloor =cellingfloor+ double(endcell1 - begincoldet) /
    CLOCKS_PER_SEC;
#endif

//void Bresenham3D(Vector3d *x1, Vector3d *x2, double start_error[],
    int start_cell[], int end_cell[])
#ifdef runningtimecheck
clock_t beginBresenham3D=clock();
#endif

double dxa[DIM*(N-1)];
double xa[DIM*N];
Eigen::Map<MatrixXd>(dxa,DIM,N-1) = DX;
Eigen::Map<MatrixXd>(xa,DIM,N) = X;
int i, x_inc, y_inc, z_inc;
int point[DIM];
double err_1, err_2, dx2, dy2, dz2, l, m, n, Esx, Esy, Esz;

```

```
for (int ii=0; ii<N-1; ii++)
{
//Bresenham3D(ii,&Vector3d(X.col(ii)),&Vector3d(X.col(ii+1)), &
    Vector3d(NodsCellErr.col(ii)),&Vector3i(NodsCellLoc.col(ii)),&
    Vector3i(NodsCellLoc.col(ii+1)));
point[0] = NodsCellLocs[ii*DIM];
point[1] = NodsCellLocs[ii*DIM+1];
point[2] = NodsCellLocs[ii*DIM+2];
x_inc = (dxa[ii*3] < 0) ? -1 : 1;
l = abs(dxa[ii*3]);
y_inc = (dxa[ii*3+1] < 0) ? -1 : 1;
m = abs(dxa[ii*3+1]);
z_inc = (dxa[ii*3+1] < 0) ? -1 : 1;
n = abs(dxa[ii*3+2]);

dx2 = l+l;
dy2 = m+m;
dz2 = n+n;
int cellnum=incell(point[0],point[1],point[2]);
#ifdef SparseVectorCellIndex
if (CellTakenSparse.coeff(cellnum)==0)
{
CellTakenSparse.insert(cellnum)=CellBeTakenNum+1;
CellBeTaken[CellBeTakenNum]=cellnum;
CellBeTakenNum+=1;
}

SegsInCell[(CellTakenSparse.coeff(cellnum)-1)*MaxCols+SegsInCellNum[
    CellTakenSparse.coeff(cellnum)-1]]=ii;
SegsInCellNum[CellTakenSparse.coeff(cellnum)-1]+=1;
#else
```

```
if (CellTaken[cellnum]==0)
{
CellTaken[cellnum]=CellBeTakenNum+1;
CellBeTaken[CellBeTakenNum]=cellnum;
CellBeTakenNum+=1;
}

SegsInCell[(CellTaken[cellnum]-1)*MaxCols+SegsInCellNum[CellTaken[
    cellnum]-1]]=ii;
SegsInCellNum[CellTaken[cellnum]-1]+=1;
#endif
// record the information of the mid part of the segment
if ((l >= m) && (l >= n)) {
Esy=y_inc*(2*NodsCellErrs[ii*3+1]-1)*l-x_inc*(2*NodsCellErrs[ii*3]-1)
    *m;
Esz=z_inc*(2*NodsCellErrs[ii*3+2]-1)*l-x_inc*(2*NodsCellErrs[ii*3]-1)
    *n;
//Esx=2*es*l;
err_1 = Esy+ dy2 - l;
err_2 = Esz+ dz2 - l;
// not recording the location of start and end points of the segment
// because it has already been calculated in the start_cell and
    end_cell
// so i<abs(end_cell[0]-start_cell[0])-1
for (i = 1; i < abs(NodsCellLocs[(ii+1)*3]-NodsCellLocs[ii*3]); i++)
{
  if (err_1 > 0) {
    point[1] += y_inc;
    err_1 -= dx2;
  }
  if (err_2 > 0) {
    point[2] += z_inc;
    err_2 -= dx2;
  }
  err_1 += dy2;
  err_2 += dz2;
  point[0] += x_inc;

  //cout<<begin(SegsCellLocs)<<endl;
  //cout<<endl<<point[0]<<" "<<point[1]<<" "<<point[2]<<" "<<endl;
  //cout<<endl<<SegsCellLoc[segi].locs[loctemp]<<" "<<SegsCellLoc[
    segi].locs[loctemp+1]<<" "<<SegsCellLoc[segi].locs[loctemp+2]<<"
    "<<endl;

  //new version
#ifdef SparseVectorCellIndex
  cellnum=incell(point[0],point[1],point[2]);
  if (CellTakenSparse.coeff(cellnum)==0)
  {
    CellTakenSparse.insert(cellnum)=CellBeTakenNum+1;
    CellBeTaken[CellBeTakenNum]=cellnum;
    CellBeTakenNum+=1;
  }

  SegsInCell[(CellTakenSparse.coeff(cellnum)-1)*MaxCols+SegsInCellNum
    [CellTakenSparse.coeff(cellnum)-1]]=ii;
  SegsInCellNum[CellTakenSparse.coeff(cellnum)-1]+=1;

#else
  if (CellTaken[incell(point[0],point[1],point[2])]==0)
  {
    CellTaken[incell(point[0],point[1],point[2])]=CellBeTakenNum+1;
    CellBeTaken[CellBeTakenNum]=incell(point[0],point[1],point[2]);
    CellBeTakenNum+=1;
  }
  SegsInCell[(CellTaken[incell(point[0],point[1],point[2])]-1)*
    MaxCols+SegsInCellNum[CellTaken[incell(point[0],point[1],point
    [2])]-1]]=ii;
  SegsInCellNum[CellTaken[incell(point[0],point[1],point[2])]-1]+=1;
#endif
}
}
else if ((m >= l) && (m >= n)) {
Esx=x_inc*(2*NodsCellErrs[ii*3]   -1)*m-y_inc*(2*NodsCellErrs[ii*
     3+1]-1)*l;
Esz=z_inc*(2*NodsCellErrs[ii*3+2]-1)*m-y_inc*(2*NodsCellErrs[ii*
     3+1]-1)*n;
err_1 = Esx+ dx2 - m;
err_2 = Esz+ dz2 - m;
for (i = 1; i < abs(NodsCellLocs[(ii+1)*3+1]-NodsCellLocs[ii*3+1]) ;
     i++) {
  //output->getTileAt(point[0], point[1], point[2])->setSymbol(symbol
    );
  if (err_1 > 0) {
    point[0] += x_inc;
    err_1 -= dy2;
  }
  if (err_2 > 0) {
    point[2] += z_inc;
    err_2 -= dy2;
  }
  err_1 += dx2;
  err_2 += dz2;
```

```
  point[1] += y_inc;

  //new version
#ifdef SparseVectorCellIndex
  cellnum=incell(point[0],point[1],point[2]);
  if (CellTakenSparse.coeff(cellnum)==0)
  {
    CellTakenSparse.insert(cellnum)=CellBeTakenNum+1;
    CellBeTaken[CellBeTakenNum]=cellnum;
    CellBeTakenNum+=1;
  }

  SegsInCell[(CellTakenSparse.coeff(cellnum)-1)*MaxCols+SegsInCellNum
    [CellTakenSparse.coeff(cellnum)-1]]=ii;
  SegsInCellNum[CellTakenSparse.coeff(cellnum)-1]+=1;

#else
  if (CellTaken[incell(point[0],point[1],point[2])]==0)
  {
    CellTaken[incell(point[0],point[1],point[2])]=CellBeTakenNum+1;
    CellBeTaken[CellBeTakenNum]=incell(point[0],point[1],point[2]);
    CellBeTakenNum+=1;
  }
  SegsInCell[(CellTaken[incell(point[0],point[1],point[2])]-1)*
   MaxCols+SegsInCellNum[CellTaken[incell(point[0],point[1],point
    [2])]-1]]=ii;
  SegsInCellNum[CellTaken[incell(point[0],point[1],point[2])]-1]+=1;
#endif
}
} else {
Esy=y_inc*(2*NodsCellErrs[ii*3+1]-1)*n-z_inc*(2*NodsCellErrs[ii*
    3+2]-1)*m;
Esx=x_inc*(2*NodsCellErrs[ii*3]   -1)*n-z_inc*(2*NodsCellErrs[ii*
    3+2]-1)*l;
err_1 = Esy+dy2 - n;
err_2 = Esx+dx2 - n;
for (i = 1; i < abs(NodsCellLocs[(ii+1)*3+2]-NodsCellLocs[ii*3+2]) ;
    i++) {
  //output->getTileAt(point[0], point[1], point[2])->setSymbol(symbol
    );
  if (err_1 > 0) {
    point[1] += y_inc;
    err_1 -= dz2;
  }
  if (err_2 > 0) {
    point[0] += x_inc;
    err_2 -= dz2;
  }
  err_1 += dy2;
  err_2 += dx2;
  point[2] += z_inc;

  //new version
#ifdef SparseVectorCellIndex
  cellnum=incell(point[0],point[1],point[2]);
  if (CellTakenSparse.coeff(cellnum)==0)
  {
    CellTakenSparse.insert(cellnum)=CellBeTakenNum+1;
    CellBeTaken[CellBeTakenNum]=cellnum;
    CellBeTakenNum+=1;
  }

```

```
    SegsInCell[(CellTakenSparse.coeff(cellnum)-1)*MaxCols+SegsInCellNum
      [CellTakenSparse.coeff(cellnum)-1]]=ii;
    SegsInCellNum[CellTakenSparse.coeff(cellnum)-1]+=1;

#else
    if (CellTaken[incell(point[0],point[1],point[2])]==0)
    {
       CellTaken[incell(point[0],point[1],point[2])]=CellBeTakenNum+1;
       CellBeTaken[CellBeTakenNum]=incell(point[0],point[1],point[2]);
       CellBeTakenNum+=1;
    }
    SegsInCell[(CellTaken[incell(point[0],point[1],point[2])]-1)*
      MaxCols+SegsInCellNum[CellTaken[incell(point[0],point[1],point
      [2])]-1]]=ii;
    SegsInCellNum[CellTaken[incell(point[0],point[1],point[2])]-1]+=1;
#endif
}
}

point[0] = NodsCellLocs[ii*3+3];
point[1] = NodsCellLocs[ii*3+4];
point[2] = NodsCellLocs[ii*3+5];

//new version
#ifdef SparseVectorCellIndex
cellnum=incell(point[0],point[1],point[2]);
if (CellTakenSparse.coeff(cellnum)==0)
{
CellTakenSparse.insert(cellnum)=CellBeTakenNum+1;
CellBeTaken[CellBeTakenNum]=cellnum;
CellBeTakenNum+=1;
```

```
}

SegsInCell[(CellTakenSparse.coeff(cellnum)-1)*MaxCols+SegsInCellNum[
    CellTakenSparse.coeff(cellnum)-1]]=ii;
SegsInCellNum[CellTakenSparse.coeff(cellnum)-1]+=1;

#else
if (CellTaken[incell(point[0],point[1],point[2])]==0)
{
CellTaken[incell(point[0],point[1],point[2])]=CellBeTakenNum+1;
CellBeTaken[CellBeTakenNum]=incell(point[0],point[1],point[2]);
CellBeTakenNum+=1;
}
int celllllll=CellTaken[incell(point[0],point[1],point[2])]-1;
int testt=(CellTaken[incell(point[0],point[1],point[2])]-1)*MaxCols+
    SegsInCellNum[CellTaken[incell(point[0],point[1],point[2])]-1];
SegsInCell[(CellTaken[incell(point[0],point[1],point[2])]-1)*MaxCols+
    SegsInCellNum[CellTaken[incell(point[0],point[1],point[2])]-1]]=
    ii;
SegsInCellNum[CellTaken[incell(point[0],point[1],point[2])]-1]+=1;
#endif
double test=0.0;
}

#ifdef SparseVectorCellIndex
CellTakenSparse.finalize();
#endif

#ifdef runningtimecheck
clock_t endBresenham3D=clock();
Bresenham3Dtime=Bresenham3Dtime+ double(endBresenham3D -
    beginBresenham3D) / CLOCKS_PER_SEC;
#endif
//------------------------      celling end   --------------------//

//double tempx[3*N];
//Eigen::Map<MatrixXd>(tempx,3,N) = X;
MatrixXd tempx = X;
for (int ii=0;ii<CellBeTakenNum; ii++)
{
int tempcellloc=CellBeTaken[ii];
int tepnum=SegsInCellNum[ii];
// check the segments in current cell
if (tepnum>1)
{
for (int jj=0; jj<tepnum; jj++)
{
int segjj=SegsInCell[ii*MaxCols+jj];
for (int kk=jj+1;kk<tepnum;kk++)
{
int segkk=SegsInCell[ii*MaxCols+kk];
if (ColCheckFlg[segjj*(N-1)+segkk]==0)
//if (abs(segjj-segkk))
{
  //cout<<"check collision in the self cell"<<endl;
  double distance=0.0;
#ifdef runningtimecheck
clock_t begindets=clock();
#endif
  //int collision_flag=dist3D_Line_to_Line(&Vector3d(tempx.col(segjj)
    ),&Vector3d(tempx.col(segjj+1)),&Vector3d(tempx.col(segkk)),&
    Vector3d(tempx.col(segkk+1)),&distance);
```

```
int status=5;
double c1=0;
double c2=0;

Vector3d u=Vector3d(DX.col(segjj));
Vector3d v=DX.col(segkk);
Vector3d w=tempx.col(segjj)-tempx.col(segkk);

//double a=u.dot(u);
double a=lsqu;
double b=u.dot(v);
//double c=v.dot(v);
double c=lsqu;
double d=u.dot(w);
double e=v.dot(w);

double D=a*c-b*b;

if (D<(Error*a))
{
  status=1;
  c1=0.0;
  if (b>=c)
  { c2=d/b; }
  else
  { c2=e/c; }
}
else
{
  status=0;
  c1 = (b*e - c*d)/D;
```

```
      c2 = (a*e - b*d)/D;
    }

    if ((c1<=0)||(c1>=1)||(c2<=0)||(c2>=1))
    {
      status=2;
    }
    distance = (w + (c1 * u) - (c2 * v)).norm();
#ifdef runningtimecheck
clock_t endcell2=clock();
coldecting =coldecting+ double(endcell2 - begindets) / CLOCKS_PER_SEC
    ;
#endif
    if (status==0)
    //if (collision_flag==0)
    {
      if (distance<=DeltDis)
      {
        SelfColSta[segjj*(N-1)+segkk]=1;
        SelfColSta[segkk*(N-1)+segjj]=1;
      }
      else
      {
        ColCheckFlg[segjj*(N-1)+segkk]=1;
        ColCheckFlg[segkk*(N-1)+segjj]=1;
      }
    }
    else
    {
      ColCheckFlg[segjj*(N-1)+segkk]=1;
      ColCheckFlg[segkk*(N-1)+segjj]=1;
```

```
  }
}}}
}

//check its neighbour cells
for (int jj=0;jj<tepnum;jj++)
{
int segjj=SegsInCell[ii*MaxCols+jj];;
for (int kk=0;kk<13;kk++)
{
int temcelllocneighbour=tempcellloc+CellNeighbours[kk];
if(temcelllocneighbour<0)
{
  cout<<"Cell is too small"<<endl;
  system("pause");
}
#ifdef SparseVectorCellIndex
if (CellTakenSparse.coeff(temcelllocneighbour)>0)

{
  for(int ll=0;ll<SegsInCellNum[CellTakenSparse.coeff(
    temcelllocneighbour)-1];ll++)

  {
    int segkk=SegsInCell[(CellTakenSparse.coeff(temcelllocneighbour)
    -1)*MaxCols+ll];

#else
if (CellTaken[temcelllocneighbour]>0)

{
```

```
for(int ll=0;ll<SegsInCellNum[CellTaken[temcelllocneighbour]-1];ll++)

{
int segkk=SegsInCell[(CellTaken[temcelllocneighbour]-1)*MaxCols+ll];
#endif
if (ColCheckFlg[segjj*(N-1)+segkk]==0)
{
  double distance=0.0;
#ifdef runningtimecheck
clock_t begindets=clock();
#endif
  //int collision_flag=dist3D_Line_to_Line(&Vector3d(tempx.col(segjj)
    ),&Vector3d(tempx.col(segjj+1)),&Vector3d(tempx.col(segkk)),&
    Vector3d(tempx.col(segkk+1)),&distance);
  int status=5;
  double c1=0;
  double c2=0;

  Vector3d u=Vector3d(DX.col(segjj));
  Vector3d v=DX.col(segkk);
  Vector3d w=tempx.col(segjj)-tempx.col(segkk);

  //double a=u.dot(u);
  double a=lsqu;
  double b=u.dot(v);
  //double c=v.dot(v);
  double c=lsqu;
  double d=u.dot(w);
  double e=v.dot(w);

  double D=a*c-b*b;

  if (D<(Error*a))
  {
    status=1;
    c1=0.0;
    if (b>=c)
    { c2=d/b; }
    else
    { c2=e/c; }
  }
  else
  {
    status=0;
    c1 = (b*e - c*d)/D;
    c2 = (a*e - b*d)/D;
  }

  if ((c1<=0)||(c1>=1)||(c2<=0)||(c2>=1))
  {
    status=2;
  }
  distance = (w + (c1 * u) - (c2 * v)).norm();

#ifdef runningtimecheck
clock_t endcell2=clock();
coldecting =coldecting+ double(endcell2 - begindets) / CLOCKS_PER_SEC
    ;
#endif


  //if (collision_flag==0)
```

```
    if (status==0)
    {
        if (distance<=DeltDis)
        {
            SelfColSta[segjj*(N-1)+segkk]=1;
            SelfColSta[segkk*(N-1)+segjj]=1;
        }
        else
        {
            ColCheckFlg[segjj*(N-1)+segkk]=1;
            ColCheckFlg[segkk*(N-1)+segjj]=1;
        }
    }
    else
    {
        ColCheckFlg[segjj*(N-1)+segkk]=1;
        ColCheckFlg[segkk*(N-1)+segjj]=1;
    }
}}}

}
}
}
#ifdef drawcellbetaken
drawcellnumber=CellBeTakenNum;
copy(begin(CellBeTaken),end(CellBeTaken),begin(drawcell));
#endif
return 0;
}
```

codes/CollisionDetection.h

```
void Distance (Vector3d *L1Start,Vector3d *L1End, Vector3d *L2Start,
    Vector3d *L2End, Vector3d *distance, double *c1, double *c2)
{
// the direction of distance is from the points on L2 to L1
// status: 0: the distance is between L1,and L2
//          1: L1 is parallel to L2
//          2: the distance is outside L1,or L2
// currently cannot deal with the thread in the same line

  Vector3d u=*L1End-*L1Start;
  Vector3d v=*L2End-*L2Start;
  Vector3d w=*L1Start-*L2Start;

  double a=u.dot(u);
  double b=u.dot(v);
  double c=v.dot(v);
  double d=u.dot(w);
  double e=v.dot(w);

  double D=a*c-b*b;
  *c1 = (b*e - c*d)/D;
  *c2 = (a*e - b*d)/D;

  *distance = w + (*c1 * u) - (*c2 * v);
}



//**** continuous penalty forces for line to line
void CPF_L2L(Vector3d *CPF, Vector3d *P0,Vector3d *Q0,Vector3d *R0,
    Vector3d *S0,Vector3d *VP, Vector3d *VQ, Vector3d *VR, Vector3d *
    VS, double dt, double dl, double spring_constant, int sign)
```

```
{// P:the start point of line segment L1; Q: the end point of L1;
// R:the start point of line segment L2; S: the end point of L2;
// VP,VQ,VR,VS are the speed of the four end points of L1,L2
// the direction of penalty force is: L2—>L1
Vector3d d10=*Q0-*P0;
Vector3d d20=*S0-*R0;
Vector3d w0=*P0-*R0;

double a0=d10.dot(d10);
double b0=d10.dot(d20);
double c0=d20.dot(d20);
double d0=d10.dot(w0);
double e0=d20.dot(w0);

double WP=1-(b0*e0-c0*d0)/(a0*c0-b0*b0);
double WQ=(b0*e0-c0*d0)/(a0*c0-b0*b0);
double WR=1-(a0*e0-b0*d0)/(a0*c0-b0*b0);
double WS=(a0*e0-b0*d0)/(a0*c0-b0*b0);

Vector3d n0p=(*Q0-*P0).cross(*R0-*S0);
Vector3d n1p=(*VQ-*VP).cross(*R0-*S0)+(*Q0-*P0).cross(*VR-*VS);
Vector3d n2p=(*VQ-*VP).cross(*VR-*VS);
double n0n=n0p.norm();
Vector3d n0=n0p/n0n;
Vector3d n1=n1p/n0n;
Vector3d n2=n2p/n0n;

// integration for penalty moment
double t=0;
Vector3d Wc0=WP*(*P0+*VP*t)+WQ*(*Q0+*VQ*t)-WR*(*R0+*VR*t)-WS*(*S0+*VS
    *t);
```

```
Vector3d nwc0=sign*Wc0/Wc0.norm();
Wc0=dl*nwc0-sign*Wc0;
Vector3d NE0=n0+n1*t+n2*t*t;
t=dt;
Vector3d Wc=WP*(*P0+*VP*t)+WQ*(*Q0+*VQ*t)-WR*(*R0+*VR*t)-WS*(*S0+*VS*
    t);
Vector3d nwc=sign*Wc/Wc.norm();
Wc=dl*nwc-sign*Wc;
Vector3d NE=n0+n1*t+n2*t*t;
Vector3d I=spring_constant*(NE0*Wc0.transpose()*NE0+NE*Wc.transpose()
    *NE)*dt/2;

*CPF=I/dt;
}
// variables in the function: CollisionResponse() start
Vector3d Fcon=Vector3d::Zero(); //contact force
Vector3d dist=Vector3d::Zero();
Vector3d Fc11=Vector3d::Zero();
Vector3d Fc12=Vector3d::Zero();
Vector3d Fc21=Vector3d::Zero();
Vector3d Fc22=Vector3d::Zero();
double c1=0;
double c2=0;
// variables in the function: CollisionResponse() end
int CollisionResponse()
{
for (int ii=0; ii<=N-4; ii++)
{
  for (int jj=ii+2; jj<=N-2; jj++)
  {
  //if (SELFCSS(ii,jj)==1)
  if (SelfColSta[ii*(N-1)+jj]==1)
  {
    CPF_L2L(&Fcon,& Vector3d(X.col(ii)),& Vector3d(X.col(ii+1)),&
   Vector3d(X.col(jj)),&Vector3d(X.col(jj+1)),
              & Vector3d(V.col(ii)),& Vector3d(V.col(ii+1)),&Vector3d(V
   .col(jj)),&Vector3d(V.col(jj+1)),h,DeltDis,SpringConst,1);

    Distance(&Vector3d(X.col(ii)),&Vector3d(X.col(ii+1)),&Vector3d(X.
   col(jj)),&Vector3d(X.col(jj+1)),&dist, &c1, &c2);
    Fc11=Fcon*(1-c1);
    Fc12=Fcon-Fc11;
    Fc21=-1*Fcon*(1-c2);
    Fc22=-1*Fcon-Fc21;

    Fext.col(ii)=Fext.col(ii)+Fc11;
    Fext.col(ii+1)=Fext.col(ii+1)+Fc12;
    Fext.col(jj)=Fext.col(jj)+Fc21;
    Fext.col(jj+1)=Fext.col(jj+1)+Fc22;
  }
  }
}
return 0;
}
```

codes/CollisionResponse.h

```
// variables in the function: UpdatePosition() start
// employ Picard iteration to update the positions of each node
VectorXd Lambda=VectorXd::Zero(N-1);    // Lagrangian multipliers
VectorXd Lambdanew=VectorXd::Zero(N-1);    // Lagrangian multipliers

MatrixXd DV=MatrixXd::Zero(DIM,N-1);
MatrixXd DFext=MatrixXd::Zero(DIM,N-1);
MatrixXd DXBar=MatrixXd::Zero(DIM,N-1);

MatrixXd EE=MatrixXd::Zero(DIM,DIM);
Vector3d Lamdbablock=Vector3d::Zero();
MatrixXd E1N=MatrixXd::Zero(DIM,DIM-1);
MatrixXd LEX=MatrixXd::Zero(DIM,N-1);

VectorXd Rhs=VectorXd::Zero(N-1);
#ifdef SparseMatrixSolver
SpMat LHS(N-1,N-1);
#ifdef ParallelComputing
tbb::concurrent_vector<T> coefficients;
#else
  std::vector<T> coefficients; // list of non-zeros coefficients
#endif
#else

MatrixXd LHS=MatrixXd::Zero(N-1,N-1);
#endif



// variables in the function: UpdatePosition() end
void UpdatePosition()
{
DV=V.block(0,1,DIM,N-1)-V.block(0,0,DIM,N-1);
DFext=Fext.block(0,1,DIM,N-1)-Fext.block(0,0,DIM,N-1);
DXBar=DX+h*DV+(hsqu/m)*DFext;

double err=1.0;
int Count=1;
```

```
#ifdef runningtimecheck
  clock_t start=clock();
#endif
//LEX=EX;

Rhs.array()=lsqu-(DXBar.array()*DXBar.array()).colwise().sum().
    transpose();

Vector2d templhs2;
Vector3d templhs;
while ((err>100*Error)&&(Count<MaxIter))
{


#ifdef SparseMatrixSolver
  E1N.col(0)=-EX.col(0);
  E1N.col(1)=EX.col(1);
  templhs2=(2*DXBar.col(0)+E1N*(Vector2d(Lambda(0),Lambda(1)))).
    transpose()*E1N;
  coefficients.push_back(T(0,0,templhs2(0)));
  coefficients.push_back(T(0,1,templhs2(1)));
  //LHS.coeffRef(0,0)=templhs2(0);
  //LHS.coeffRef(0,1)=templhs2(1);
#ifdef ParallelComputing
  tbb::parallel_for(1,N-2, [=](int ii) //(int ii=1; ii<=N-3; ii++)
  {
    Matrix3d ET;
    ET.col(0)=EX.col(ii-1);
    ET.col(1)=-2*EX.col(ii);
    ET.col(2)=EX.col(ii+1);
    Vector3d templhs1=((2*DXBar.col(ii)+(ET*Vector3d(Lambda(ii-1),
     Lambda(ii),Lambda(ii+1)))).transpose()*ET);
      coefficients.push_back(T(ii,ii-1,templhs1(0)));
      coefficients.push_back(T(ii,ii,templhs1(1)));
      coefficients.push_back(T(ii,ii+1,templhs1(2)));

   });
#else
   for (int ii=1; ii<=N-3; ii++)
   {
     EE.col(0)=EX.col(ii-1);
     EE.col(1)=-2*EX.col(ii);
     EE.col(2)=EX.col(ii+1);
     Lamdbablock=Vector3d(Lambda(ii-1),Lambda(ii),Lambda(ii+1));
     templhs=(2*DXBar.col(ii)+(EE*Lamdbablock)).transpose()*EE;
      coefficients.push_back(T(ii,ii-1,templhs(0)));
      coefficients.push_back(T(ii,ii,templhs(1)));
      coefficients.push_back(T(ii,ii+1,templhs(2)));
   }
#endif
   E1N.col(0)=EX.col(N-3);
   E1N.col(1)=-2*EX.col(N-2);
   templhs2=(2*DXBar.col(N-2)+E1N*Vector2d(Lambda(N-3),Lambda(N-2))).
     transpose()*E1N;
   coefficients.push_back(T(N-2,N-3,templhs2(0)));
   coefficients.push_back(T(N-2,N-2,templhs2(1)));
   LHS.setFromTriplets(coefficients.begin(), coefficients.end());
   coefficients.clear();

#ifdef runningtimecheck
   clock_t start1=clock();
#endif
```

```
  Eigen::SparseLU<SparseMatrix<double> >  SparseLUsolver;
  SparseLUsolver.compute(LHS);
  SparseLUsolver.analyzePattern(LHS);
  SparseLUsolver.factorize(LHS);
  if(SparseLUsolver.info()!=Success) {
    // decomposition failed
    cout<<"Decomposition failed"<<endl;
    system("pause");
    exit(0);
  }
  Lambdanew = SparseLUsolver.solve(Rhs);

#ifdef runningtimecheck
  clock_t end1=clock();
  solvingeqtime=solvingeqtime+double(end1 - start1) / CLOCKS_PER_SEC;
#endif
#else
  E1N.col(0)=-EX.col(0);
  E1N.col(1)=EX.col(1);
  LHS.block(0,0,1,2)=(2*DXBar.col(0)+E1N*(Vector2d(Lambda(0),Lambda
    (1)))).transpose()*E1N;
  for (int ii=1; ii<=N-3; ii++)
  {
    EE.col(0)=EX.col(ii-1);
    EE.col(1)=-2*EX.col(ii);
    EE.col(2)=EX.col(ii+1);

    LHS.block(ii,ii-1,1,3)=(2*DXBar.col(ii)+EE*Vector3d(Lambda(ii-1),
   Lambda(ii),Lambda(ii+1))).transpose()*EE;
  }
  E1N.col(0)=EX.col(N-3);
```

```
    E1N.col(1)=-2*EX.col(N-2);
    LHS.block(N-2,N-3,1,2)=(2*DXBar.col(N-2)+E1N*Vector2d(Lambda(N-3),
      Lambda(N-2))).transpose()*E1N;
#ifdef runningtimecheck
    clock_t start1=clock();
#endif
    Lambdanew=LHS.ldlt().solve(Rhs);
#ifdef runningtimecheck
    clock_t end1=clock();
    solvingeqtime=solvingeqtime+double(end1 - start1) / CLOCKS_PER_SEC;
#endif
#endif
    err=(Lambda-Lambdanew).array().abs().maxCoeff();
    Lambda=Lambdanew;

    Count+=1;
}
 if (Count>=MaxIter)
{
    cout<<"Cannot converge for the Pircard iteration! Possible reason:
      the time step is too large. Try to decrease it."<<endl;
    system("pause");
}


#ifdef runningtimecheck
    clock_t end=clock();
    picardtime=picardtime+double(end - start) / CLOCKS_PER_SEC;
#endif

 for (int ii=1;ii<=N-2;ii++)
```

```
{
   V.col(ii)=V.col(ii)+h/m*Fext.col(ii)+hRec*(Lambda(ii)*EX.col(ii)-
     Lambda(ii-1)*EX.col(ii-1));
}
V.col(N-1)=V.col(N-1)+h/m*Fext.col(N-1)-hRec*Lambda(N-2)*EX.col(N-2);

X.block(0,1,3,N-1)=X.block(0,1,3,N-1)+h*V.block(0,1,3,N-1);
}
```

codes/UpdatePosition.h