DISSERTATION
submitted
to the
Combined Faculties for the Natural Sciences and for
Mathematics
of the
Ruperto-Carola University of Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

Put forward by
Diplom-Phys. Nathan Hüsken
born in Landau (Pfalz)

Oral examination:

Realtime Simulation of Stiff Threads for microsurgery training simulation

Advisor: Prof. Dr. Reinhard Männer

for my family and friends

Abstract

This thesis introduces the physical simulation of surgical thread for usage in a microsurgical training simulator for the education of medical students. To allow interactive simulation the thread must be real time capable. Importantly, in the simulation, the thread must behave in a way that it looks like a real thread to the user. The user can then "dive into" the simulation, because for the user, the simulation of the thread appears real. We refer to this "diving into" the simulation as "immersion". The physical model of the thread is a mass-spring model based on the Kirchhoff theory for elastic rods. One challenge is the stiffness constraint of the thread. A real world thread does not change it's length significantly even under high stress. In a mass-spring model this property can be obtained by using high spring constants. But if an explicit integration method is applied the so called "overshooting" effect presents a problem. It causes the system to diverge when the spring constants are too high. In this thesis the problem is addressed by applying an implicit integration method. A key property of implicit integration methods is that it is unconditionally stable and thereby allows a large time step in the numerical integration. But it also requires that a linear system of size equal to the number of degrees of freedom in the system is solved. If the number of degrees of freedom is high this conflicts with the real-time requirement of the simulation. In this work it is shown that for the case of the thread the matrix in the linear system is banded and can therefore be solved in linear time. Another advantage of the implicit integration is that forces are propagated along the complete thread within one time step.

For the simulation of microsurgical sutures knots have to be modeled. A knot causes numerous contacts of the thread with itself. The contact forces are modeled and calculated using a physical model. Because all forces propagate along the whole thread within one time step all contacts interact with each other. A force applied at one contact affects all other contacts.

Because of this all contact forces have to be solved for simultaneously. The interaction of the contacts due to the implicit integration are calculated resulting in a linear system which describes the relation between the contact forces and the relative movement of the thread at the contacts. Physically correct contact forces have to be found with this linear system. Similar to the simulation of rigid bodies, a linear complementary problem or a nonlinear complementary problem results depending on the model that is used for the contact forces. In case of rigid body simulation the "projected Gauss-Seidel" is a proven method for solving the problem. In this thesis the nonlinear conjugate gradient (NNCG) method from Silcowitz-Hansen et al. [101] is applied. This method was originally developed for rigid body simulations.

The thread has been integrated into the microsurgical training simulator "MicroSim". Which is to say, interactions between the thread and tissue and forceps have been modeled and incorporated into "MicroSim". These interactions have to be compatible with the implicit integration of the thread. In a joint work with Sismanidis [102] and Schuppe [96] a training module for MicroSim has been developed. This training module allows for training of a microsurgical anastomosis of blood vessels.

Zusammenfassung

In dieser Arbeit wird eine physikalische Simulation eines chirurgischen Fadens für den Einsatz in einem mikrochirurgischen Trainingssimulator zur Ausbildung von Medizinern vorgestellt. Der Faden muss echtzeitfähig sein, um in einer interaktiven Simulation eingesetzt werden zu können. Um die Immersion der Simulation zu garantieren muss das Verhalten des Fadens glaubhaft sein. Der Faden wird mit einem auf der Kirchhoff Theorie für elastische Stäbe basierenden Feder-Masse Modell simuliert. Eine Herausforderung stellt dabei die Zugfestigkeit des Fadens da. Ein realer Faden verändert seine Länge selbst unter starkem Zug kaum. In einem Feder-Masse Modell kann diese Eigenschaft durch hohe Federkonstanten abgebildet werden. Bei Verwendung eines expliziten Integrationsverfahren führen hohe Federkonstanten zu dem sogenannten "overshooting" Effekt, bei dem das System divergiert. In dieser Arbeit wird das Problem durch Anwendung eines impliziten Integrationsverfahren gelöst. Implizite Integrationsverfahren zeichnen sich dadurch aus, dass sie unbedingt stabil und dadurch einen hohen Zeitschritt bei der numerischen Integration erlauben. Sie beinhalten allerdings auch, dass ein lineares Gleichungssystem von der Größe der Anzahl der Freiheitsgrade in jedem Zeitschritt gelöst werden muss. Für Systeme mit vielen Freiheitsgraden steht der hierfür benötigte Rechenaufwand im Widerspruch zu der Echtzeitanforderung der Simulation. Für den Fall des Fadens wird in dieser Arbeit gezeigt, dass die Matrix in dem Gleichungssystem eine Bandmatrix ist. Dadurch lässt das Gleichungssystem sich in linearer Zeit lösen. Die implizite Integration hat auch den Vorteil, dass Kräfte innerhalb eines Simulationsschrittes entlang des gesamten Fadens propagieren.

Um die Simulation von mikrochirurgischen Nähten zu ermöglichen müssen mit dem simulierten Faden Knoten geformt werden können. In einem Knoten gibt es viele Kontakte des Fadens mit sich selber. Die Kontaktkräfte werden physikalisch modelliert und berechnet. Da Kräfte innerhalb eines Simulationsschrittes zwischen den Kontaktpunkten propagieren, interagieren alle Kontakte miteinander. Wird an einem Kontakt eine Kraft angewendet, verändert dies die Voraussetzungen an allen anderen.

Dies bedeutet, dass die Kontaktkräfte für alle Kontakte simultan gefunden werden müssen. Dafür werden die Interaktion der Kontakte durch die implizite Integration berechnet. Das Ergebnis ist ein Gleichungssystem für die Relation zwischen den Kontaktkräfte und die relativen Verschiebung an den Kontakten. Für dieses Gleichungssystem müssen die physikalisch korrekten Kontaktkräfte gefunden werden. Ähnlich wie bei der Simulation von starren Körpern ergibt sich, je nach Formulierung der Voraussetzungen an die Kontaktkräfte, ein lineares Komplementaritätsproblem (engl. linear complementarity problem, kurz LCP) oder ein nichtlineares Komplementaritätsproblem (engl. non-linear complementarity problem, kurz NCP). Bei der Simulation von starren Körpern ist eine bewährte Methode das "projected Gaus-Seidel" (kurz PGS) Verfahren. In dieser Arbeit wird die nonlinear nonsmooth conjugate gradient (NNCG) Methode von Silcowitz-Hansen et al. [101], welche auch für Starrkörpersimulation entwickelt wurde, angewendet.

Der Faden wurde in den mikrochirurgischen Trainingssimulator "MicroSim" eingebunden. Dafür wurden Interaktionen, welche mit der impliziten Integration kompatibel sind, mit dem Gewebe und den Pinzetten modelliert. In Zusammenarbeit mit Sismanidis [102] und Schuppe [96] wurde ein Trainingsmodul für MicroSim entwickelt, welches das Trainieren einer mikrochirurgischen Anastomosis von Blutgefäßen erlaubt.

Acknowledgement

I would like to thank . . .
. . . Prof. Dr. Reinhard Männer for his support as my supervisor.
. . . the management of the VRmagic GmbH for supporting me and the MicroSim project.
. . . my colleagues of the ViPA group, namely Florian Beier, Oliver Schuppe and Evangelos Sismanidis, for the great conversations and time we had together.
. . . all the people of VRmagic, which I will not name separately to spare me the embarrassment of forgetting someone, who supported me whenever I needed it.
. . . my mother for to many things to list here.
. . . my friends Mélanie and Annika for finding the last grammatical and spelling errors (except for those which will hopefully never be found).
. . . and my step father for looking through and correcting the thesis in detail. Thank you Stuart!

Contents

List of Figures

List of Tables

1. Introduction

Microsurgery is a subdiscipline of surgery which is characterized by the use of an operating or surgical microscope.

In 1921, a pioneer of microsurgery – *C. O. Nylén* – used a monocular microscope for experiments with the ears of rabbits. In the same year he performed operations on the human ear using a binocular microscope.

Since then microsurgery has made tremendous advancements. In 1964, Harry Buncke made the first replantation of a rabbit's ear. Buncke's success has been seen as a milestone because the replantation required an anastomosis of blood vessels of only $1mm$ in diameter. This is the size of the main blood vessels supplying muscles and skin. With the continuous improvement of microscopy, microsurgery became more and more important in medicine[1].

Today, microsurgery has many applications in various medical fields, such as rhinology, ophthalmology and neurosurgery. Anastomosis of blood vessels and nerves in the millimeter and sub-millimeter scale remains an important discipline especially for the free transfer of organs and the replantation of tissue.

To protect the patients' safety a sound education and training of surgeons is required. Because of the small scale in which the operations are performed in microsurgery, the surgeon needs a steady hand and precise coordination. This is further complicated by the indirect work with the microscope. One can only indirectly see the actions performed with one's hand. Hence a good hand-eye coordination must be acquired. Moreover, the surgeon must have a well-developed confidence in his skills so that he can react well in stressful and unexpected situations. The needed skills are best acquired through direct and frequent exercise of the procedures.

A common procedure in the education of medical students is the "See one, do one, teach one" approach. A student watches operations and thereby learns how to perform them. As Mason and Strike [71] observed, this is still the most commonly practiced method of training medical students. Bergamaschi [12] argues that this approach is not appropriate anymore considering the complexity and required skills for medical tasks and operations today.

To avoid training surgeons directly on patients, a variety of simple models, manikins, animals and cadavers are used [40, 123, 46]. All these methods have their advantages and disadvantages:

[1]Miehlke [72] describes the history of microsurgery.

- **Manikins** can portray the anatomy of the patient and the steps of the procedure can be illustrated visually. Low tech manikins are inexpensive but not suited for training of complex and filigree procedures.
- Training on **animals** has the advantage that one operates on real tissue. However the use of animals for microsurgery training raises ethical and legal concerns. In addition, access to laboratory animals is not always possible[40]. Furthermore, animal anatomy differs from human anatomy. Therefore an animal can only be seen as an approximation of a human[123].
- Human **cadavers** have the correct anatomy, but dead tissue responds different from live tissue. In addition the acquisition of cadavers is costly and limited. Additional ethical and legal questions have also to be considered with the usage of dead bodies.

Virtual reality (VR) simulations are therefore increasingly in the focus of attention when considering the training of medical students. There are several works supporting the importance and usage of surgical training simulators [39, 81, 55].

With a simulator a medical student can practice common scenarios. But also unusual situations which seldomly occur in real operations can be simulated, thus allowing the student to be prepared for them when they occur in the operation room. The simulation can be set up and switched to a different scenario quickly without a lengthy assembly of the needed parts or costly acquisitions. This allows for higher numbers of students being trained and for more flexibility without the need of a tutor watching and supervising the training. Importantly, no ethical or legal problems with animals, cadavers or patient safety occur.

Training simulators avoid many issues occurring with other methods. In addition, the issue of error management is another advantage of VR simulations as a training method, as Ziv et al. [123] argue:

> "Simulation offers ethical benefits, increased precision and relevance of training and competency assessment, and new methods of teaching error management and safety culture."

Ziv et al. argue that medical simulations have much to offer in the regard to error management, training for risky procedures, and assessing competences. They further argue

> "[Medical simulation] reduces the exposure of patients to health professionals that are less experienced, and thus contributes to better protection of patient rights to receive quality care that focuses on the patient's needs."

Virtual reality simulations are well suited for the training of many medical procedures. Similar to pilots who train for unforeseen as well as common scenarios in a simulator, surgeons can train for both unanticipated and common situations. In 2012, Sigounas et al. [99] investigated the effect of three vascular surgery simulators on the advancement of surgical skills of surgeons. Their work showed a significant improvement of skills for the test persons.

1.1. The MicroSim project

MicroSim is a microsurgery simulator which is developed by the ViPA[2] group of the "Lehrstuhl für Informatik V" under the supervision of *Prof. Dr. R. Männer* in cooperation with the *VRmagic GmbH.* A prototype has been published in [56].

The goal of the MicroSim project is the creation of a microsurgical training simulator which supports students in the acquisition and improvement of microsurgery skills. Trainees learn indirect hand-eye coordination while working with a microscope and learn to deal with tiredness and tremor due to unusual positioning of hands and arms. It allows students to train regularly without a time consuming setup and thereby supports early learning success. It also avoids the ethically questionable use of animals, human cadavers or endangering of patients.

In MicroSim the display is a stereo display which is used in such as wat as the way as in a real surgery. Trainees use real microsurgical forceps to perform the virtual interventions. Lannon et al. [65] argues:

> Microsurgery lends itself well to immersion because both the real and virtual environments are observed via an intermediate optical device with a relatively narrow field of vision.

He further argues that in contrast to other surgical disciplines no haptic feedback is needed due to the small scale of the operation. The use of real instruments increases the immersion of the system.

For tracking, colored markers are applied to the forceps. Cameras track the markers allowing the reconstruction of the forceps' position and opening. The tracking space is inside a box laid out with black velvet. The black velvet allows the camera to better distinguish the markers from the background. The tracking system has been developed by Schuppe [96].

MicroSim is developed to support multiple training scenarios which can be switched according to the trainee's needs.

[2]Virtual Patient Analysis

1.2. Goal of the Thesis

A difficult task in microsurgery is the anastomosis of blood vessels. The filigree thread requires a steady hand and good hand-eye coordination. The tying of knots requires a degree of advanced spatial imagination. The goal of this thesis is the development of a simulation of a thread to be used in the microsurgery simulator MicroSim. Real time capability is a requirement of the simulation.

The thread has to interact with the tissue and the forceps. It should be able to collide with both objects. The trainee must be able to grab the thread with the forceps. When the thread gets into a configuration where it exerts force on the tissue, the tissue has to react. A suture consists of the two points where the thread goes through the tissues to be joined, in such a way that when the tread is tied, it pulls these two tissues (or blood vessels) together.

The thread also has to interact with itself. It must come in contact with itself and the trainee must be able to tie knots. Knots should hold tight and later dissolve realistically in accordance with to behavior of the same knot in the real world.

Even more important than physical realistic behavior is that the topology of the configuration does not change. This means that the thread must not pass through itself or tunnel through tissue or the forceps. When the thread passes through itself it could potentially cause a knot to dissolve. Also when it passes through tissue or the forceps, unrealistic configurations can be created.

The thread should be adjustable in it's behavior. It will be most beneficial if the adjustment is directly related to the visual features of its behavior. For example parameters that directly control the strength of bending and torsion forces.

1.3. Contributions

Real time simulation of a stiff thread with implicit integration The Kirchhoff theory for elastic rods [64] has been used for the deformation model of the thread.

For a sustainable training of suturing, the simulation of the thread must be realistic. Unrealistic simulation of the thread can lead the student to develop inappropriate or faulty habits. A major difficulty with the simulation of threads is that a real thread is very stiff along it's longitudinal axis. While it is a deformable object it's behavior when pulled along it's longitudinal axis is more like that of a rigid body. If one pulls at one end the force is transferred to the other end without stretching the thread significantly. On the other hand, bending and twisting behaviors show the characteristics of a deformable object. When using a force based model, as it seems appropriate for a deformable object, forces have to propagate fast along the thread without causing numerical instabilities.

A formalism for an implicit integration scheme has been developed and applied to a discretization of a Kirchhoff rod. The implicit integration allows for making the thread stiff. It is shown that the resulting linear system is solvable in linear time[3] making the simulation realtime capable. For this all forces and their spatial derivatives based on the discretized of a Kirchhoff rod have been derived. For the torsional forces the formalism from Bergou et al. [13] has been used.

Contact handling and simulation of knots Due to the implicit integration all contacts involving the thread interact with each other. The interactions of the contacts have been quantified. This allows for solving all contacts at once in a global solver. A contact and friction model based on Coulomb friction has been applied and solved similarly to how contacts of rigid bodies are commonly solved. The contact model allows for simulation of knots that hold tight or dissolve in a realistic way.

Incorporation of the thread into MicroSim and building of an anastomosis training module The thread simulation has been integrated into the microsurgery simulator MicroSim. Contact handling with the tissue and forceps that is compatible with the implicit integration of the thread has been developed. The thread can also be grabbed by the forceps enforcing the position of the grabbed part. A training module in which the tying of a knot for a microsurgery end-to-end anastomosis has been developed. The module trains anastomosis of blood vessels as it occurs e. g. in a *free DIEP* breast reconstruction. The trainee has to knot and secure a knot holding together two blood vessels. The simulation of the blood vessels has been developed by Sismanidis [102].

[3]linear in the number of vertices the simulated thread is made of

2. Related Work

2.1. Simulation of threads and rods

Simulation of threads, rods or other one dimensional objects is an active research area with applications in hair simulation, computer games and medical virtual reality simulations. For the state of the art in strand and hair simulation, refer to the surveys by Ward et al. [119] and Hadap et al. [51].

One of the most simple models for thread simulation is “Follow the leader”, introduced by Brown et al. [18]. The thread is discretized into N control points. When an external constraint grasps and moves a control point, the motion is propagated along the thread, moving all control points such that the length constraints are maintained. While this approach is extremely efficient and allows hard constraining of the total length of the thread, 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.

In the work by Müller et al. [78] “Follow the leader” is extended with “Dynamic Follow-The-Leader” for hair simulation allowing a more dynamic behavior of the hair strands.

Not directly related but similar is the work of Kubiak et al. [61]. In this work the authors employ the position based approach from Müller et al. [77] for constraints. This results in an algorithm similar to “Follow the leader” with torsion and bending. A torsion constraint governs the torsional torques. The algorithm is unconditionally stable but to archive stiff bending or torsion, many iterations per integration step are needed.

For a more immerse behavior one can look at increased physically motivated approaches. Physically-based simulation and animation of deformable objects is an active research area in computer graphics. For an overview of the topic refer to [79].

Physically based models for threads include **spline** based approaches. In these models the rod or thread is also discretized into vertices. The position of the vertices represent the control points of a spline curve.

The earliest spline-based deformation model known to the author has been proposed by Terzopoulos et al. [110]. They modeled elastically deformable objects using physically based dynamics of splines. While they focus on a general model for elastically deformable objects, one of the examples they mention is a telephone cord.

A spline contains more information than a linear connection of the control points. The local curvature (bending) and torsion are also represented by this mathematical description. Qin and Terzopoulos [90] take advantage of this fact by formulating Lagrangian mechanics for the equation of motion of D-NURBS solving them with finite element analysis. Rémion et al. [91] use a similar technique. The Lagrangian formalism is applied to obtain the dynamic movement of a spline curve to model knitted clothing.

Using this work as a base, Lenoir et al. [67] creates a real-time performance thread with the goal of using it to simulate a surgical thread. They are able to create knots by handling self-collisions with penalty methods. While this allows to preserve the topological properties of the thread, physical behavior is not represented accurately enough, making knots breakable. Also, torsional behavior is not included.

Another simpler but still physically motivated approach is the **mass spring** model. Similar to the spline approach a spatial discretization into mass points is carried out. The mass points are connected by springs enforcing the stretching constraint and with more complex springs possible other constraints.

The first mass spring model for thread-like objects known to the author of this thesis was introduced by Rosenblum et al. [92] to simulate hair. To be able to simulate large quantities of hair, they chose a very simple model of mass points connected by springs. Additional hinge springs provide bending resistance.

Anjyo et al. [5] also used a physically based approach to simulate hair, based on the mass spring model. A lot of heuristics based on aesthetic observations are applied, which allows faster processing than the direct physical formulation would allow. The resulting elastic laws result in a particular simple second-order differential equation with an analytically derived solution for the dynamics of the mass points. The same deformation model has been used by Daldegan et al. [27], developing a framework for the simulation of hair. Plante et al. [86] and Chang et al. [21] focus on simulating cluster of hairs. This reduces the complexity of simulating large amounts of hair strains and simplifies collision detection.

Phillips et al. [85] do knot-tying with a spring-based deformation model. Their model is adaptive allowing for the insertion and removing of mass points as required by the configuration of the thread. Knots are not maintained by the modeling of friction or explicit constraints but simple impulse based collision resolution. Often the holding of knots is not correctly simulated.

Wang et al. [117] add torsional behavior to threads simulated with a mass spring system. The control points are linked with torsional springs. This requires an additional degree of freedom describing the torsion of the thread. The whole system is integrated with a simple Euler integrator. On the other hand Selle et al. [97] attach triangles to the edges of a hair strand. The triangles allow capturing twist and add springs which counteract the twist.

A common starting point for physically based simulation of rods or threads which includes torsion behavior is the **Kirchhoff theory of elastic rods**. For details see the work of Langer and Singer [64]. Yang et al. [122] employed finite elements to solve the discretized equations obtained from the theory. Klapper [60] applied finite differences for the same purpose.

Goldstein and Langer [45] observed that using the **Bishop frame** simplifies both the analytical formulation and the numerical implementation of the dynamics of symmetric rods especially when considering the behavior under twist. The Bishop frame has been used in most of the later works involving twisting behavior of threads or rods. As this work also utilizes the Bishop frame, it is explained in more detail in section 3.1.1.

Another physical theory used in many simulation models for threads and rods is the **Cosserat** deformation model. It is inspired by the Kirchhoff theory [57]. As in the Kirchhoff theory, the idea of the model is an oriented continuous curve with an orthonormal basis at every point corresponding to the material frame. Stress, and thereby forces are described by spatial differentiation of the frame. The main drawback of most Cosserat models is that they reconstruct the center line of the thread instead of explicitly simulating it. This makes it difficult to find and maintain contacts or other geometric constraints. With spline-based deformation models, mass spring models or other models with an explicit center line representation, contacts are much easier to maintain. The main advantage of the Cosserat based models is its consistent handling of torsion.

Pai [83] first introduced the Cosserat in the field of computer simulation. Using the Cosserat theory, differential equations are derived which allow for finding the configuration from boundary conditions. These boundary conditions are given by the orientations and positions of the end points of a solid object, such as a thread. In other words, the explicit representation of the thread between those boundary points is not stored but reconstructed in every simulation step. This complicates the handling of collisions and makes preservation of the topology very difficult.

Bertails et al. [14] provide a robust solution for simulating the dynamics of a Cosserat rod in the field of hair simulation. Their approach allows bending and twisting instabilities as well as buckling of hair stands. A drawback is that the mass-matrix is dense which lets the number of operations grow with $O(N^2)$ per time step (where N is the number of segments).

Grégoire and Schömer [48] model bending and torsion following the Cosserat model with a generalized mass-spring system. They chose quaternions to represent the angular orientation of segments deriving the forces from the representation. This requires coupling the quaternion with the positional degrees of freedom. Quaternions are also used in the work of Choe et al. [23]. They introduce a framework for simulating the dynamic movement of human hairs treating it as a collection of wisps. The segments of a wisp are represented as rigid bodies with a center of mass and an orientation represented by quaternions. The bodies are connected by linear and angular springs. A similar approach is used by Green et al. [47] where quaternions are also used to represent the orientations of segments. But instead of the center of mass, the end points (which connect the segments) are used for the representation. To couple the quaternions to the segment orientation a penalty method is used.

Spillmann and Teschner [104] address the contact handling problems most Cosserat models have due to the lack of an explicit representation for the center line. They introduce a deformation model called "CoRdE". They utilize finite elements and compute energies per element, instead of solving the boundary problem as other Cosserat inspired models do. Later they extend their approach in [105] with a method for adaptively subdividing the rod depending on the local bending, allowing them to build knots with short segments while not reducing performance with a high number of vertices. Punak and Kurenov [89] simplifies "CoRdE" to archive better performance. The CoRdE model can bee seen as a mass-spring approach where forces are based on the Cosserat theory.

A different representation of the center line is used in the work of Wakamatsu and Hirai [116]. Instead of representing the center line by explicit vertex positions, it is parametrized by the Euler angle between the material frames making them the degrees of freedom. They introduce a modeling approach for linear objects based on an extension of differential geometry. The energy is minimized to obtain the configuration of a deformed rod. Similar to Pai [83] this approach does only allow for computation of static deformation. Theetten et al. [113] have proposed an extension of their deformation model that can dynamically switch between a static and a dynamic solution, depending on the required simulation context.

Hadap [50] introduces an oriented strands system, which is implemented as the dynamics of a serial multi body chain. He uses an implicit center line representation based on reduced curvature coordinates. The chain is made very stiff by employing an implicit integration method formulated as a linear complementarity problem (LCP).The LCP is then solved with an iterative scheme. While this allows for simulation of very stiff strands, the LCP solver is computationally too complex to allow real-time simulation.

The work of Theetten et al. [112] has combined the Cosserat theory with a spline based deformation model. Each vertex gains an additional degree of freedom: It's rotation field. The twisting force can be expressed as the sum of the geometric twisting and the material torsion, which is given by the rotation field. The model allows for very accurate simulation even of stiff objects utilizing a fast implicit integration method from Hilde et al. [53]. Their approach allows for dynamic behavior and is geometrically exact where the only approximation is the spatial and temporal discretization. They achieve reasonable interactive simulation rates except in the case of very high stiffness, for which the integration method requires excessive computation time for stability.

Another approach to rod or thread simulation is **chain shape matching** (CSM), see for example the work of Rungjiratananon et al. [93]. The rod is represented by a chain of particles which are connected by segments. **Chain regions**, which are multiple overlapping, group the particles. External forces move the particles independently. For each region (which are treated as rigid), variables like rotation and translation are computed with the new particle positions. The transformations of the regions define goal positions for the particles. The goal position for each particle is obtained from the average of all regions of which it is a part of. The stiffness is controlled by the size of the regions. Collisions are resolved by moving penetrating segments apart. This allows simulating stiff threads with self collisions. But physical realistic knots with friction are difficult to model due to the nonphysical nature of this approach.

Bergou et al. [13] introduce a reduced coordinate formulation with a minimal number of degrees of freedom avoiding the need of stiff coupling between material frame and center line. The torsion of the material is represented by the torsion angle between the untwisted and the current configuration, which is stored with every segment. The problem of propagating this torsion angle is avoided by observing that twist waves propagate much faster than other material deformations (like bending). Motivated by this observation, they set the torsion angle in such a way that the energy is evenly distributed along the thread. For a thread without intrinsic bending, this reduces to distributing the difference between neighbored torsion angles evenly along the thread. This approach is also applied in this thesis. The twist forces are derived from the spatial derivative of the material torsion using discrete holonomy. Their model reproduces both static and dynamic phenomena. The model for torsional behavior in this thesis is greatly inspired by this work.

Other recent works are [70] and [66]. Martin et al. [70] create a unified treatment for elastic rods, shells and solids. Such an approach misses the benefit of incorporating the properties unique to a thread. Larsson et al. [66] allow the thread to have arbitrary cross sections. Their simulation is based on Continuum mechanics based threads and is real time capable.

2.2. Implicit Integration

While a thread is in general a deformable object, it usually cannot be made to change it's length significantly without big forces. When modeling it as a chain of mass points connected by springs, this poses great difficulties to the integration. The model needs to be integrated forward in time. Mathematically this corresponds to solving an ordinary differential equation with a given initial value: the equation of motion. One of the simplest integration methods is the Euler integrator. Applied to a physical problem, the Euler integrator assumes that the force is constant within one time step. This is subject to error. It can lead to diverging systems in the case of stiff equations. The assumption of the constant force can add an unrealistic amount of energy to the system and transform it into a situation where the forces are even higher. This leads to a problem called "overshooting". For a more detailed explanation, refer to section 3.2.

Different integration methods, such as the Leap-frog, Runge-kutta [19], or Verlet [115] improve the stability, but are still not able to stably integrate extreme stiff equations. Incidentally, the Verlet integrator is used for the simulated tissue in the MicroSim, the microsurgery simulator the thread is made for.

A simple method for avoiding these errors is reducing the time step with which the model is simulated. Unfortunately this approach hits the limit of what is computationally possible with modern computers when simulating stiff threads. But this problem is not inherent to the simulations of threads. When trying to simulate cloth with almost inextensible fabric, the same problem occurs. Terzopoulos et al. [110][108][109] use implicit integration schemes to simulate various deformable objects.

The idea of implicit integration is simple: Instead of using the force at the current time step, the force at the next time step is applied [88, chapter 17.5]. This archives unconditional stability which allows integrating stiff springs allowing to model the thread with springs. Implicit integration has the disadvantage that a linear system of equations has to be solved in every time step. For a more in depth explanation refer to section 3.2.

The fact that implicit integration is unconditionally stable allows for increasing the time step and thereby reducing the computational burden. Also, forces are distributed throughout the complete system within one time step. Baraff and Witkin [10] demonstrate that implicit methods for cloth simulation can overcome the performance limits inherent in explicit simulation methods. The linear system is solved with a fast iterative solution algorithm, a modified version of the conjugate gradient (CG) method.

Servin and Lacoursière [98] also use a model made of constrained rigid bodies connected in a chain. With this model they simulate a cable that can hold a heavy load. The physics are integrated with an implicit scheme. The occurring linear equations are solved with a Gauss Seidel approach.

Sobottka et al. [103] use the implicit integration scheme from Chung and Hulbert [24] called the Generalized-α method to integrate the dynamic Cosserat equations.

Desbrun et al. [30] propose an implicit integration method that allows fast integration of arbitrary deformable objects with a mass spring model. The computational burden of solving the linear system is reduced by splitting forces into linear and nonlinear components. This results in the linear problem of the implicit integration being also split into two, one for the linear forces and one for the nonlinear forces. For the linear part the matrix is constant, which means that it can be solved in a pre-computation step by inverting the matrix. For the nonlinear part it is assumed that the forces are constant within one time step, as with an explicit integration scheme. This way the matrix is zero.

Other works ensure the stiffness of the thread by explicitly enforcing the non extensibility of the thread. Bergou et al. [13] apply the fast manifold projection from [44]. First an explicit symplectic Euler integrator is applied to integrate the unconstrained system. Then the manifold projection calculates a nearby constraint configuration.

A similar approach has been done by Becker [11]. With a projector the external forces are projected onto all mass points. The resulting forces do not stretch the thread. Due to linearization errors a correction step, correcting the length of the segments is necessary after every integration step. Allner [3] also distributes external forces, but directly without a projector following a Gauss distribution around the mass point where the force is applied.

2.3. Collision and Contact Handling

For a realistic simulation of a suturing situation, especially for the simulation of knots, it is crucial that collisions of the thread with itself are resolved in a physical way. Finding the interpenetration of the thread with itself and other objects is a geometrical task and will be handled in sections 3.5, 3.6 and 3.7.

Once contacts have been found, they must be resolved in such a way that any interpenetration of objects are removed. Furthermore, a realistic behavior of knots which correspondents to their behavior in the real world is required. This can be archived with contact and friction forces taken from a physical model.

One of the more simple approaches for resolving collisions are **penalty forces**. The general idea is to apply a force on penetrating objects in the direction of separation. A common choice for the magnitude of this force is making it proportional to the penetration depth. Setting a good proportional constant requires fine tuning. If the constant is too high, objects get pushed apart beyond separation. If it is too low objects continue to penetrate up to the point where the separation direction is no longer well defined. In addition, for many situations, a universal proportion constant does not exist. For example, when two boxes are stacked on each other more force is needed to separate the lower box from the ground than when only one box is lying on the ground. Works employing penalty forces are [59, 29].

The example with the stacked boxes is in the realm of rigid bodies. In the realm of deformable bodies, penalty forces are more suitable. The reason is that when deformable objects collide, there is a non zero contact time [84]. In one integration step the objects get compressed by the penalty force and it takes one or more integration steps to propagate that compression to other colliding objects. On the other hand, for rigid objects the change in momentum is instantaneous and therefore should propagate to all other colliding objects within the same integration step.

Penalty forces allow for penetration of the objects which can lead to ambiguities for small objects. Terzopoulos et al. [110] cope with this problem in the area of hair simulation by putting a force field around hair strands which is much bigger than the hair strand itself to prevent collision. This can be seen as preemptive penalty forces, which act before the objects collide.

Gissler et al. [43] improve penalty forces by setting the proportionality constant per collisions to a value which separates the (deformable) objects exactly in the next integration step. This avoids the tuning of the proportional constant and results in a more concise penalty force. Yet, for rigid body situations, where contacts interact, this does not help to prevent penetrations in stacking situations.

Penalty forces resolve the problem locally for every collision. Different from approaches which try to find global solutions, this has a low computational burden. But for situations in which contacts interact – such as stacking of rigid objects – it is impossible to find a correct solution locally. This is because the question of how big the separating force has to be does not only depend on the objects in contact but for example also on whether objects are stacked on top.

To completely prevent the penetration of objects in the first place, **predictor-corrector** approaches separate the integration in a predictor and a projector step. In the predictor step, normal integration takes place and penetrations are detected. Then the integration step is rewound and forces preventing the detected penetrations are added to the system. The correction step than repeats the integration with the added forces included. Weinstein et al. [121] do this for rigid bodies by computing allowable trajectories before moving the rigid bodies to their new positions. For rods, Spillmann and Teschner [104] employ a predictor-corrector method integrated with an explicit integration method. Guendelman et al. [49] use a predictor-corrector scheme for the collisions for massive amounts of rigid objects. They proposed employing the predicted positions to compute the impulses. However, since the resting contacts require more accuracy, they use the original positions to compute the contact forces.

Outside of the realm of force based approaches, **position-based** approaches resolve collision and contact situations by explicitly setting the positions of colliding objects such that interpenetrations are resolved. This requires the simulation to be suited for this, e. g. being position based itself. Also for rigid objects contacts can still interact. Moving two objects apart can result in one of the objects penetrating a new object. Milenkovic [73] apply linear programming to find positions that yield the non-penetration constraint. The linear programming is necessary to guarantee that these constraints are fulfilled for all objects at the same time. Müller et al. [77] bases its position-based approach on the velocity verlet integrator. Since they focus on deformable objects no global solver is needed.

To ensure consistent collision responses and contact handling for situations where contacts interact, such as stacking of rigid bodies, some sort of **global solver** is needed. A global solver solves all contact forces at once. If one ignores friction forces and solves only the penetration problem, the problem can be formulated as a linear complementary problem (short LCP, see [32]), as it is done in [106] and [20]. Catto [20] also includes a simplified friction model by which the problem remains a LCP problem. The Coulomb friction model states that the magnitude of the frictional force F_{fric} is limited by $F_{fric} \leq \mu F_N$ where μ is the coefficients of friction and F_N the magnitude of the normal force. Catto [20] simplifies this constraint by $F_{fric} \leq \mu'$ with some constant μ'. While by this simplification the problem remains an LCP problem, it leads to some unrealistic situations. For example in a stack of boxes the friction on the lowest box is the same as the friction on boxes located above it.

The full constraint $F_{fric} \leq \mu F_N$ leads to the friction force being restricted to a circle. If one draws this circle in dependents of F_N the so called "friction cone" results. Stewart and Trinkle [106] and Anitescu and Potra [4] approximate this cone by a polyhedral cone. This way the problem remains a LCP.

The contact problem with the correct Coulomb friction formulation, with the restriction that the coefficient of friction for kinetic and static friction are equal, can be formulated as a nonlinear complementary problem (NCP), which has the advantage of a lower memory consumption than the LCP formulation with a polyhedral friction cone ([101, 106]).

The formulation of the NCP for the contact problem with the thread will be done in section 3.7.2.

In many situations, if one knew the nature of the situation in advance, the problem would be much less complex. An example of such a situation is a stack of boxes. One could propagate an impulse from the lowest box upward respecting the constraint, that the box below cannot be accelerated downwards. The idea from Hahn [52] is similar. They built a contact graph. Impulses are propagated starting from immovable nodes. This ensures that no immovable nodes are penetrated. For example a simple stack of objects is solved by propagating from the ground to the top box. When the graph has cycles, contacts have to be rechecked after one iteration of traversing the graph. Guendelman et al. [49] built on this idea. They noticed that only strongly connected components in the contact graph have to be solved at once, which they do with an iterative method. They start with infinite mass objects (immovable objects) and propagate into strongly connected components. Afterward the masses of all objects in the components are temporally set to infinite and the propagation is continued. The contact graph allows them to efficiently propagate shocks.

For the thread in this thesis, this would not yield any improvements because all contacts interact and therefor only one strongly connected component, involving all contacts, exists.

The contact problem can also be formulated as a quadratic programming (QP) problem (see e. g. [74]). Milenkovic and Schmidl [74] improve their previous work in [73]. Stacking is done with standard Newtonian physics and an optimization based method to adjust correcting the predicted positions of objects to avoid overlaps. Contacts, collisions and position updates are solved with quadratic programming. Kaufman et al. [58] reduce the expensive global QP problem for a velocity-level simulation to two convex, separable QPs per object. Different methods for formulating this QP for bodies composed of polyhedral and curved surfaces have been presented in [6, 7, 69, 41].

Schmidl [95] improve on their previous work in [74] by freezing objects that are still, or almost still, removing the need to solve constraints for them.

Once the LCP or NCP has been formulated, it has to be solved. An often used direct solver is **Lemkes algorithm** [8, 106]. Lloyd [68] improve Lemkes algorithm for rigid body simulations lowering the computational burden from $O(n^3)$ to $O(mn+m^2)$ where n is the number of contacts and m the number of bodies in the system.

Different from the direct methods, an iterative solver improves the solution by applying the same step iteratively. An iterative solver for optimization problems is the conjugate gradient method. It has been reported that the conjugate gradient method is computationally intractable due to frequent changes of the active set [38]. The active set is the set of contacts for which the contact force is non zero. Additional iterative methods are the Newton and the Interior point methods. In theory the Newton method converges quadratically, but Lacoursiere [62] reports linear convergence rates in experiments. A step in the Newton method requires solving a linear subsystem and has therefor a complexity of $O(n^3)$. Even when only few steps are required the complexity of the Newton method is cubic. For the interior-point method the same problem occurs as these methods require solving a linear subsystem in every step too.

A simple iterative method is the projected Gauss-Seidel (PGS) approach, which is explained e. g. by Silcowitz-Hansen et al. [101]. The PGS loops over all contacts solving the current contact. Catto [20] applied it to rigid body physics with a simplified friction model. Duriez et al. [31] applies a Gauss-Seidel like method to solve the frictionless contact problem for deformable objects simulated with finite element methods. Another work utilizing PGS is [38].

Most iterative methods, like PGS, have the advantage that they can be stopped at any time and give an approximation of the correct solution. This approximation improves with each iteration step. In contrast direct methods like Lemkes algorithm cannot yield any useful results before they have finished. This allows for limiting the computational burden of these iterative methods by stopping when the consumed time exceeds a given threshold. In addition, they benefit from being initialized with an approximation of the correct solution. For the approximation the result from the last simulation step can be taken. The method of initializing a solver with an approximation of the correct result is called “warm-starting” [20].

Poulsen et al. [87] and Silcowitz-Hansen et al. [100] indicate that the PGS method is well suited for solving NCP problems for contact forces, as it is robust and versatile. Based on this experience Silcowitz-Hansen et al. [101] improve the convergence rate of the PGS method by a Fletcher-Reeves type nonlinear nonsmooth conjugate gradient (NNCG) type method. A PGS step is interpreted as the gradient of a nonsmooth nonlinear quasi-quadratic function. The local minimizer of the squared PGS step (which corresponds to the error) is found with a Fletcher-Reeves nonlinear conjugate gradient method. The NNCG method has been used with a small adaption for contact resolution of the thread.

2.4. Microsurgery Training Simulators

One of the first training simulators for a surgical procedure known to the author was introduced by Salvendy and Pilitsis [94]. It was an electromechanical training simulator utilized as part of a study regarding skill acquisition. However, computers were not yet fast enough for computer based virtual reality training for physical simulation.

The general interest in this research can be seen for example in the numerous papers developing algorithms for microsurgery training simulators (see e. g. [17]). A particular interest is on the simulation of tissue [76, 28, 25, 15, 75].

A review for microsurgery simulators from 2003 can be found in [36].

O’Toole et al. [82] developed an interactive virtual reality surgical training simulator for the training of suturing technique. It utilizes surgical tools with force feedback by a pair of PHANToM devices. A PHANToM device is a haptic input device by SenseAble[1]. Tissue and tools are simulated with a physically based simulation. The software is able to measure and evaluate the trainee’s performance. The system is able to distinguish between experienced and novice trainees. The output is done by a 3-dimensional graphics display. The graphics are projected on semi-reflective mirrors and viewed with a pair of stereo goggles. A disadvantage of the system is that the trainee has to maintain a standing position.

Webster et al. [120] developed a training simulator for sutures. As with most simulators, it uses a haptic feedback device. It allows for training piercing and tightening of sutures, but no knot tying or other interaction with suture material.

A microsurgery simulator developed by Brown et al. [16] used two real surgical forceps tracked electromagnetically as an input device. The simulation allowed interactions between the surgical tools and the tissue modeled by deformable objects using mass-spring systems. It utilizes the “Follow the leader” approach [18] for

[1] `http://sensable.com`

thread simulation. The used collision detection and response algorithm allows for maintaining the correct topology of knots. However, it does not comply to real world situations in simulating the physical behavior for holding the knot. Also “Follow the leader” does not provide the realistic physical properties needed in a suturing situation.

Another medical simulator based on “Follow the leader” is presented by Figueras Sola et al. [42] focusing on laparoscopic suture training. It allows for training fine motor skills such as accurate grasping of the thread.

Holbrey et al. [54] [55] develop a suturing simulator for vascular surgery. Deformable objects, such as tissue, are simulated with the finite element method (FEM). For haptical input with a pair of needle holders a PHANToM device is used. Its opening and closing is detected using a magnetic switch.

Allard et al. [2] present the open source framework SOFA. It provides numerous tools and algorithms for medical simulations.

Wang et al. [118] introduce a microsurgery training simulator which allows for making sutures. The physical model is a simple explicit integrated mass spring model. The work does not explain how they deal with the strong forces occurring in a suture.

3. Theory

One of the requirements in a complex suture situation is robust detection and resolution of contacts.

An explicit representation of the center line facilitates complex contact scenarios. Many approaches use geometric curves, such as splines, for the center line [108, 90, 91, 67]. These represent the center line with the control points of a spline. Even better suited are linear connections of the control points. Mass-spring models have this property. Mass-spring models also have the advantage of being force based. This allows for interaction with other objects with forces, which is a flexible generally applicable approach. For simulation of knots it is also necessary to have realistic friction and repulsive forces, which is hard to archive with a non-force based model. A force based approach also allows for applying implicit integration, which will be discussed later.

For virtual reality it is more important that objects behave plausibly, than that their behavior accurately follows real physics. It therefore seems attractive to follow descriptive approaches. These approaches model the behavior of the simulated object by directly recreating observed behavior from the real world. An example of a descriptive model is recreating the bending behavior of a thread with springs.

On the other hand, physical models can be based on the laws of physics or some physical theory. Physical models are more general and often work better in unforeseen situations. But in some situations they get complicated and unnecessarily increase the computational burden.

The thread in this thesis is based on a physical model. But at some points descriptive approaches are used.

3.1. Notation and representation of the thread

In this section the physical model of the simulation of the thread will be introduced. This model will be discretized making it suitable for a computer simulation. All quantities describing the configuration and dynamics of the thread will be defined.

3.1.1. Deformation model

In this section the model describing the dynamic behavior of the thread is introduced.

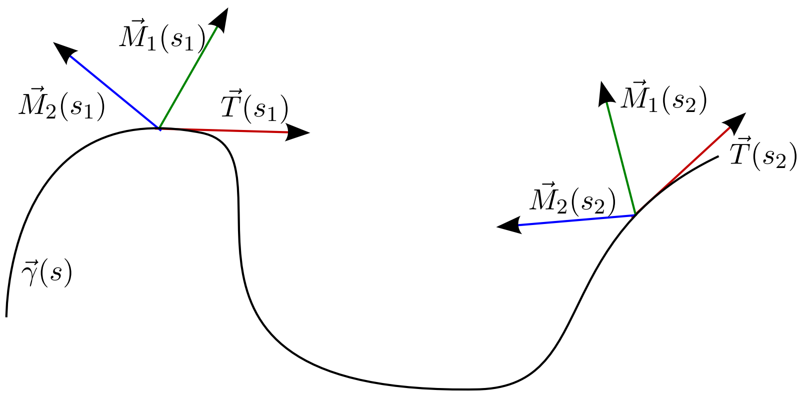


Figure 3.1.: The material frame at different positions on the thread

Many works use the Cosserat theory as a starting point for the physical model [83, 14, 47, 23]. Spillmann and Teschner [104] show that the Cosserat theory can also be adapted to an explicit center line representation. The Kirchhoff theory of elastic rods are related to the Cosserat rod model [63], but do not handle extension or shearing. Shearing is beyond the requirements of this thesis. Extension (stretching) can easily be added by adding a stretching energy. The physical model for the thread simulation used in this theory is based on the Kirchhoff theory of elastic rods[64]. The energies on which stretching, bending and torsion base are derived from the Kirchhoff theory of elastic rods. The forces are the derivatives with respect to the degrees of freedom of these energies. The following derivation is inspired by the work of Langer and Singer [64] as well as Bergou et al. [13].

Assuming the cross section of the thread does not change by elastic deformation, its configuration can be described by its center line. While in reality this assumption is of course wrong, the effect of elastic deformation of the center line on the physical behavior is so minor it can be ignored and hence does not affect immersion into the simulation.

Following the Kirchhoff theory of elastic rods, the thread is represented by an adapted frame curve $\Gamma = \{\vec{\gamma}(s); \vec{T}(s), \vec{M}_1(s), \vec{M}_2(s)\}$ over a parameter $s \in [0,1]$. $\vec{\gamma}(s) \in \mathbb{R}^3$ represents the center line of the thread. $\{\vec{T}(s), \vec{M}_1(s), \vec{M}_2(s)\}$ form an orthonormal coordinate system at every position s on $\vec{\gamma}$ called the "material frame". It is adapted to $\vec{\gamma}$ by requiring that $\vec{T}(s)$ is always aligned to $\frac{\partial}{\partial s}\vec{\gamma}(s)$ making it tangent to $\vec{\gamma}(s)$ (See figure 3.1).

For representing the position of the thread in space, $\vec{\gamma}(s)$ would suffice. But the material frame contains information about the twist of the thread. As will be seen later, the twist of the thread is represented by the rotation of $\vec{M}_1(s)$ and $\vec{M}_2(s)$ around $\vec{T}(s)$.

The Darboux Vector

The Energy of the thread can be found with this material frame using the Darboux vector which will be defined below. For this a few aspects of the material frame must be analyzed.

Because the material frame is normalized, the derivative with respect to $s \in [0,1]$ of its components must be orthogonal to the corresponding component itself. Therefore it can be represented by the corresponding other components:

$$\frac{\partial \vec{T}(s)}{\partial s} = a\vec{M}_1(s) + b\vec{M}_2(s) \tag{3.1}$$
$$\frac{\partial \vec{M}_1(s)}{\partial s} = c\vec{T}(s) + d\vec{M}_2(s) \tag{3.2}$$
$$\frac{\partial \vec{M}_2(s)}{\partial s} = e\vec{T}(s) + f\vec{M}_1(s) \tag{3.3}$$

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

Because

$$\frac{\partial}{\partial s}(\vec{T}(s) \cdot \vec{M}_1(s)) = 0 \quad \Rightarrow \quad \frac{\partial \vec{T}(s)}{\partial s}\vec{M}_1(s) = -\frac{\partial \vec{M}_1(s)}{\partial s}\vec{T}(s)$$

and equivalent calculations for the other pairs of material frame vectors, the number of unknowns in (3.1) - (3.3) can be reduced. There are ω_1, ω_2 and m such that

$$\frac{\partial \vec{T}(s)}{\partial s} = \omega_1\vec{M}_1(s) + \omega_2\vec{M}_2(s) \tag{3.4}$$
$$\frac{\partial \vec{M}_1(s)}{\partial s} = -\omega_1\vec{T}(s) + m\vec{M}_2(s) \tag{3.5}$$
$$\frac{\partial \vec{M}_2(s)}{\partial s} = -\omega_2\vec{T}(s) - m\vec{M}_1(s). \tag{3.6}$$

Defining the “Darboux vector” by $\vec{\Omega}(s) := m\vec{T}(s) - \omega_2\vec{M}_1(s) + \omega_1\vec{M}_2(s)$ allows rewriting (3.4) - (3.6) as

$$\frac{\partial \vec{T}(s)}{\partial s} = \vec{\Omega}(s) \times \vec{T}(s) \tag{3.7}$$
$$\frac{\partial \vec{M}_1(s)}{\partial s} = \vec{\Omega}(s) \times \vec{M}_1(s) \tag{3.8}$$
$$\frac{\partial \vec{M}_2(s)}{\partial s} = \vec{\Omega}(s) \times \vec{M}_2(s). \tag{3.9}$$

which will be very helpful for finding the bending and twisting energy.

Stretching energy

The stretching indicates how much the thread is stretched or compressed from it's original length. Assuming $\vec{\gamma}_0(s)$ is the unstretched configuration of the thread then it's rest length corresponds to the line lengths of $\vec{\gamma}_0(s)$. This is easily found with

$$l := \int_0^1 \left| \frac{\partial \vec{\gamma}_0(s)}{\partial s} \right| ds \tag{3.10}$$

Since $\vec{\gamma}_0(s)$ is not stretched, $\left| \frac{\partial \vec{\gamma}_0(s)}{\partial s} \right|$ must be constant and because of 3.10 equal to l.

Here it is assumed that the thread reacts to stretching and compression as a spring with spring constant k_S following Hooks law. The subscript "S" marks the constant for belonging to the stretching energy as opposed to the bending and twisting energy. The local stretching at position s is given by the difference of $\left| \frac{\partial \vec{\gamma}(s)}{\partial s} \right|$ and l. The stretch energy is given by:

$$E_S = \frac{k_S}{2} \int_0^1 \left(\frac{1}{l} \left| \frac{\partial \vec{\gamma}(s)}{\partial s} \right| - 1 \right)^2 ds \tag{3.11}$$

Assuming a spring like behavior is of course only an approximation. It has the advantage of being simple and working well with the integration algorithm used in this work. The main requirement of the stretching energy is that it does keep the thread stiff along it's main direction which is archived by using a high spring constant k_S.

Bending and Twisting Energy

The material frame is everything needed to define the bending and twisting energy of the thread. They are given by Langer and Singer [64]:

$$E_B = \frac{k_B}{2} \int_0^1 \omega_1(s)^2 + \omega_2(s)^2 ds \tag{3.12}$$

$$E_T = \frac{k_T}{2} \int_0^1 m(s)^2 ds \tag{3.13}$$

with the corresponding constants $k_B, k_T > 0$.

For an intuitive idea on how to get (3.12) and (3.13) one can picture that ω_1 and ω_2 represent how much $\vec{T}$ changes towards $\vec{M}_1$ and $\vec{M}_2$ and therefore measure how γ bends. Analog m measures the rotation of $\vec{M}_1$ and $\vec{M}_2$ around $\vec{T}$.

When the torsional energy vanishes (which implies $m = 0$), the Darboux vector reduces to

$$\vec{\Omega}_B(s) := -\omega_2(s)\vec{M}_1(s) + \omega_1(s)\vec{M}_2(s).$$

In this case (3.7) - (3.9) define a special case for the material frame, called the *Bishop frame.*

Equation (3.4) - (3.6) define how the Bishop frame evolves along the thread, but are arbitrary under the initial conditions. $\vec{T}(0)$ is given by the initial tangent, but $\vec{M}_1(0)$ and $\vec{M}_2(0)$ can be freely rotated around $\vec{T}(0)$. The consequence is, that the Bishop frame can be rotated around $\vec{T}(s)$ and still remains a Bishop frame.

Changing the initial condition adds a constant offset to the angle between the Bishop and the material frame. Since for the Bishop frame the twisting energy vanishes it defines a frame along the thread which corresponds to the most relaxed (twist-free) configuration.

With the Bishop Frame the twist of the thread can now be expressed by a simple parametrization along the thread which is more suited for deriving the discrete twisting energy later [64]. Let $\vec{M}_1^0(s), \vec{M}_2^0(s)$ be a Bishop frame and $\Phi(s)$ the angle by which the material frame is rotated around $\vec{T}(s)$ relative to the Bishop frame. It will now be shown how m is related to Φ.

$\Phi(s)$ is defined by:

$$\vec{M}_1(s) = R(\Phi(s))\vec{M}_1^0(s)$$

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

$$\frac{\partial \vec{M}_1(s)}{\partial s} = \left(\frac{\partial}{\partial s} R(\Phi(s))\right) \vec{M}_1^0(s) + R(\Phi(s))\frac{\partial \vec{M}_1^0(s)}{\partial s} \tag{3.14}$$

Recall equation (3.5). m can be extracted from it via

$$m = \frac{\partial \vec{M}_1(s)}{\partial s} \cdot \vec{M}_2(s) \tag{3.15}$$

Because the Darboux Vector for the Bishop frame has no component along $\vec{T}(s)$, $\frac{\partial \vec{M}_1^0(s)}{\partial s}$ is aligned to $\vec{T}(s)$. Therefore the second summand of (3.14) is aligned to $\vec{T}(s)$ and does not tell us anything about m. The first summand can be rewritten as:

$$\begin{aligned}\left(\frac{\partial}{\partial s}R(\Phi(s))\right)\vec{M}_1^0(s) &= \frac{\partial\Phi(s)}{\partial s}\left(\frac{\partial}{\partial\Phi}R(\Phi)\right)\vec{M}_1^0(s)\\ &= \frac{\partial\Phi(s)}{\partial s}R(\Phi(s))R(\pi/2)\vec{M}_1^0(s)\\ &= \frac{\partial\Phi(s)}{\partial s}R(\Phi(s))\vec{M}_2^0(s) = \frac{\partial\Phi(s)}{\partial s}\vec{M}_2(s) \qquad (3.16)\end{aligned}$$

To find $\frac{\partial}{\partial\Phi}R(\Phi)$ call to mind that differentiating sin and cos produces the same functions phase shifted by $\pi/2$.

From (3.15) and (3.16) it can easily be seen that

$$m = \frac{\partial\Phi(s)}{\partial s}. \qquad (3.17)$$

allowing the twisting energy to be rewritten as

$$E_T = \frac{k_T}{2}\int_0^1 \left(\frac{\partial\Phi(s)}{\partial s}\right)^2 \mathrm{d}s. \qquad (3.18)$$

This result gives an intuitive picture of the torsion energy. It is quadratic in the change of the twist angle with s.

Recall, that the Bishop frame was arbitrary under a constant rotation around $\vec{T}(s)$. But since adding a constant offset does not affect the derivative, E_T is independent of it.

3.1.2. Discretization

To make the model suitable for a computer simulation, it has to be discretized. Based on this discretizations, the forces will be calculated in section 3.3. The discretization is needed because a computer can only store a limited quantity of numbers. This means that a continuous function cannot be represented in the computer.

The discretization of stretching and bending energies is straightforward. For torsion energy the notation from Bergou et al. [13] has been adopted.

First the quantities necessary for discretization will be introduced. The center line of the thread is discretized along $\vec{\gamma}(s)$ into $N \in \mathbb{N}$ vertices. The position of the i-th vertex at time t is denoted by $\vec{p}_i$.

For reasons of readability the parameter t will be omitted where the meaning is clear, that is, without ambiguity.

Linear segments connect the vertices. The i-th segment connects the i-th and $(i+1)$-th vertex resulting in a segment vector and tangent of

$$\vec{s}_i := \vec{p}_{i+1} - \vec{p}_i \tag{3.19}$$

$$\vec{t}_i := \frac{\vec{s}_i}{|\vec{s}_i|} \tag{3.20}$$

The rest length of the $i-th$ segment is denoted by l_i.

Stretching Energy

The stretching energy $\left|\frac{\partial\vec{\gamma}(s)}{\partial s}\right|$ has to be discretized. It holds that

$$l_i = \int\limits_{L_i/l}^{L_{i+1}/l} \left|\frac{\partial\vec{\gamma}(s)}{\partial s}\right| ds$$

where $L_i := \sum\limits_{j=1}^{i} l_j$. For the discretization it is assumed that $\left|\frac{\partial\vec{\gamma}(s)}{\partial s}\right|$ is constant within a segment. Defining $\gamma_i := \left|\frac{\partial\vec{\gamma}(s)}{\partial s}\right|$. γ_i can be determined by:

$$\begin{aligned} |\vec{s}_i| &= \int\limits_{L_i/l}^{L_{i+1}/l} \left|\frac{\partial\vec{\gamma}(s)}{\partial s}\right| ds = \int\limits_{L_i/l}^{L_{i+1}/l} \gamma_i ds = \frac{l_i}{l}\gamma_i \\ \Rightarrow \quad \gamma_i &= \frac{l \cdot |\vec{s}_i|}{l_i} \end{aligned}$$

Inserting this into (3.11) reveals the discretized stretching energy.

$$\begin{aligned} E_S &= \frac{k_S}{2}\left[\int\limits_{0}^{L_2/l}\left(\frac{1}{l}\left|\frac{\partial\gamma(s)}{\partial s}\right| - 1\right)^2 ds + \cdots + \int\limits_{L_{N-2}/l}^{L_{N-1}/l}\left(\frac{1}{l}\left|\frac{\partial\gamma(s)}{\partial s}\right| - 1\right)^2 ds\right] \\ &= \frac{k_S}{2}\sum_{i=1}^{N-1}\int\limits_{L_{i-1}/l}^{L_i/l}\left(\frac{1}{l}\gamma_i - 1\right)^2 ds \\ &= \frac{k_S}{2}\sum_{i=1}^{N-1}\frac{l_i}{l}\left(\frac{|\vec{s}_i|}{l_i} - 1\right)^2 \end{aligned} \tag{3.21}$$

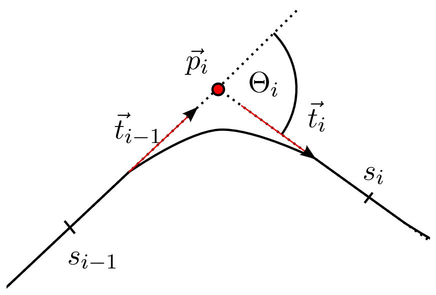


Figure 3.2.: Definition of Θ_i.

Bending Energy

Since $\frac{\partial \vec{T}(s)}{\partial s}$ is always orthogonal to $\vec{T}(s)$, using (3.4), (3.12) can be rewritten as

$$E_B = \frac{k_B}{2} \int_0^1 \left(\frac{\partial \vec{T}(s)}{\partial s} \cdot \vec{M}_1(s) \right)^2 + \left(\frac{\partial \vec{T}(s)}{\partial s} \cdot \vec{M}_2(s) \right)^2 ds = \frac{k_B}{2} \int_0^1 \left(\frac{\partial \vec{T}(s)}{\partial s} \right)^2 ds.$$

Let $\{s_i | 1 \leq i \leq N\}$ be the parameter s at the midpoints of the segments. Now $\vec{T}(s)$ is assumed to rotate around $\vec{t}_{i-1} \times \vec{t}_i$ from $\vec{T}(s_{i-1}) = \vec{t}_{i-1}$ to $\vec{T}(s_i) = \vec{t}_i$. This means, that $\left| \frac{\partial \vec{T}(s)}{\partial s} \right| = \frac{\Theta_i}{s_i - s_{i-1}}$ where Θ_i is the angle between $\vec{t}_i$ and $\vec{t}_{i-1}$ (see figure 3.2).

This reveals for the bending Energy:

$$E_B = \frac{k_B}{2} \sum_{i=2}^{N-1} \int_{s_{i-1}}^{s_i} \left(\frac{\Theta_i}{s_i - s_{i-1}} \right)^2 = \frac{k_B}{2} \sum_{i=2}^{N-1} \frac{\Theta_i^2}{s_i - s_{i-1}} = \frac{k_B}{2} \sum_{i=2}^{N-1} \frac{2l}{l_i - l_{i-1}} \Theta_i^2 \quad (3.22)$$

Note that it is assumed that before s_1 and after s_{N-1} the thread is straight and has vanishing bending energy.

Torsional Energy

Finding a discretized version for (3.13) when having (3.18) is trivial. $\Phi(s)$ is assumed to be a linear function between the segments midpoints. Defining

$$\delta\Phi_i = \Phi(s_i) - \Phi(s_{i-1}) \tag{3.23}$$

sets

$$\frac{\partial\Phi(s)}{\partial s} = \frac{\delta\Phi_i}{s_i - s_{i-1}} \qquad \text{for } s_i > s > s_{i-1}.$$

Now (3.18) can be discretized:

$$E_T = \frac{k_T}{2}\sum_{i=2}^{N-1}\int_{s_{i-1}}^{s_i}\left(\frac{\delta\Phi_i}{s_i - s_{i-1}}\right)^2 = \frac{k_T}{2}\sum_{i=2}^{N-1}\frac{2l}{l_i - l_{i-1}}\delta\Phi_i^2 \tag{3.24}$$

Choosing a segment length

The length of the individual segments has an significant impact on the performance and accuracy of the simulation. A small segment length requires the number of segments to be high and thereby increases the computational demands of the simulation. On the other hand a long segment length increases the effects of the discretization and reduced visual plausibility. When the bending between two adjacent segments is high the simulation becomes inaccurate. The forces are only forwarded along the direction of the springs. In the extreme case of a bending of 90° the adjacent springs have no common direction and no force is transmitted. But even for smaller bending angles this problem becomes noticeable.

Spillmann and Teschner [105] address this problem by splitting segments when the bending angle is too high and unsplitting them when the bending angle is low again. While this approach is applicable here, in this thesis the segment length have chosen short enough so that the problem does not occur in the first place. The reasons for this are the following:

- It is much more important that the simulation is efficient in the worst case. For the segment length this means, in the case of a lot of subdivision. And in that case there are many segments anyway. For the simulation it is irrelevant if it has to deal with many segments all the time or only in certain cases.
- As will be shown section 3.7, the limiting factor is not the number of segments but the number of contacts in a knot. So the impact on the performance is not very significant.
- While Spillmann and Teschner [105] approach conserves the energy of the thread, it still changes the circumstances for contacts. In a knot this can causes unsettle situations decreasing the stability of the knot.

3.2. Numerical integration

In this section the integration method, with which the thread is forwarded in time, is introduced. First the reason why implicit integration is necessary for this task is described. Then the integration method is derived and applied to the special case of a one dimensional object, such as a thread.

3.2.1. Numerical integration and stiff equations

A variety of numerical time integration algorithms for advancing differential equations, such as Newton's equations of motion, forward in time exists. One of the simplest is the Euler-Cromer method, also known as "last-point approximation" [26] which will serve as an example here.

Newton's equation of motion is a second order differential equation of the form

$$m \frac{\partial^2 \vec{x}}{\partial t^2}(t) = \vec{F}(\vec{x}(t)) \tag{3.25}$$

where $m > 0$ denotes the mass. Starting from given $\vec{x}(t_0), \vec{v}(t_0) := \frac{\partial \vec{x}}{\partial t}(t_0)$ the Euler method advances (3.25) from t_0 to $t_0 + \Delta t$ by

$$\begin{aligned} \vec{v}(t_0 + \Delta t) &= \vec{v}(t_0) + \frac{\vec{F}(\vec{x}(t_0))}{m} \Delta t \\ \vec{x}(t_0 + \Delta t) &= \vec{x}(t_0) + \vec{v}(t_0 + \Delta t) \Delta t. \end{aligned} \tag{3.26}$$

If (3.25) is stiff – meaning $\vec{F}(\vec{x}(t))$ is big – which happens for example when $\vec{F}(\vec{x}(t))$ resembles a stiff spring, a problem known as overshooting occurs. To illustrate the problem, imagine a one dimensional system with a mass point connected to a spring. The force on the mass point is given by the spring equation for which the rest point is assumed to be at $x = 0$ (see figure 3.3).

$$F_{Spring}(x) = -k \cdot x \tag{3.27}$$

$k > 0$ is the spring constant. The assumption (3.26) makes is, that $\vec{F}$ is constant within one time step of length Δt. If k is big this may result in a new position of the mass point, that skipped the zero point of the spring and is now further away from it (on the other side) than the original point. Because the force at this new point is even bigger the same thing happens again making the mass point diverge away from the zero point in oscillating behavior (see figure 3.4).

In the real world a thread is almost inextensible. The stretching of the thread in this thesis is modeled with springs. To make the thread very stiff, k^S has to be chosen very high. This causes overshooting, as described above.

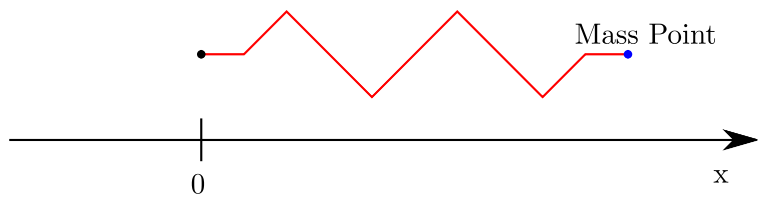


Figure 3.3.: A mass point accelerated by a spring.

A solution to this problem would be decreasing Δt up to the point where x would not cross the zero point of the spring within one time step. While this works, it is extremely computationally demanding and not feasible for real time applications. Another common approach is introducing a damping term which slows down the movement of the mass points reducing the energy of the system. This has the disadvantage that it makes the whole system behave sluggishly. For such high spring forces, which are needed to make the thread seem inextensible, the damping forces would also have to be unacceptably high.

The Euler method is a first order method, meaning it's error is of order $O(\Delta t)$. Other integration methods have an error of lower order. Among the most prominent are the verlet method [115], the more numerically stable velocity verlet [107] and the Runge-kutta method. There is also the class of semi-implicit integration methods such as the symplectic Eulor method. These methods have the advantage of better preserving the energy and thereby being more stable. For the thread to appear almost inextensible the k of the springs must be so high, that all these methods become unstable.

Niiranen [80] gives a more detailed analysis of the stability of the symplectic Eulor method.

There is another problem which becomes eminent in the case of a long chain of elements such as the discretized thread: Forces only affect neighboring elements within one integration step. A force applied to one end of the thread with N elements will affect the element on the other end after N integration steps. In the simulation many of such interactions are needed for an equilibrium to be found. For these reason using an explicit integrator causes the simulation to only converge very slowly into stable situation such as knots. The implicit integration introduced in the next section addresses these problems.

3.2.2. Implicit Integration

.

To enforce the length constraint of the thread Bergou et al. [13] project the thread to a configuration that is similar to the current configuration where the constraint

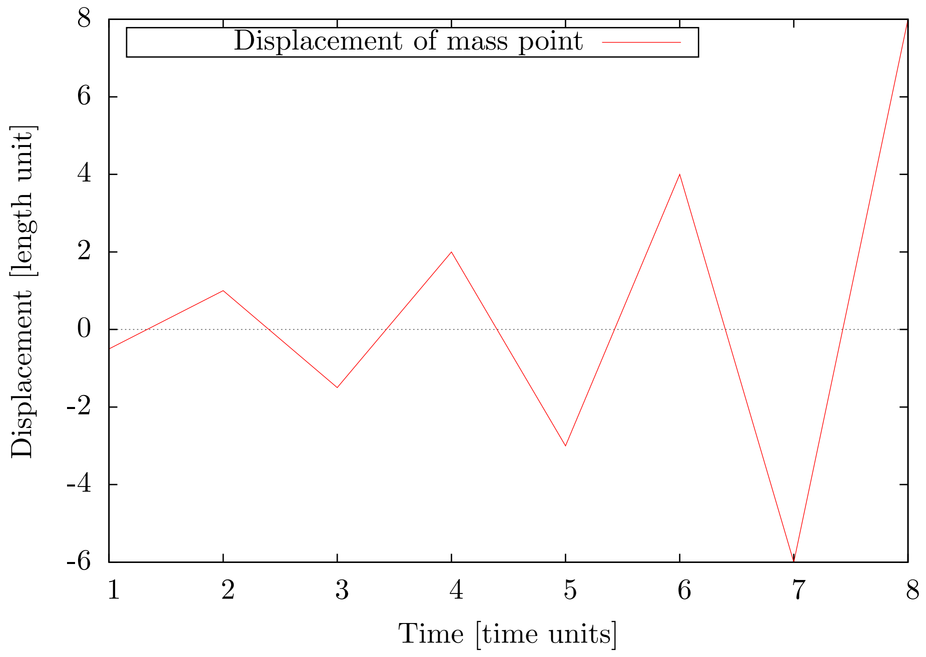


Figure 3.4.: Overshooting of a one dimensional system with a stiff spring. The displacement of the mass point from the springs rest position results in an oscillation behavior in time.

is fulfilled exactly after every integration step. The "Follow the leader" approach from Brown et al. [18] enforces the constraint by moving the vertices towards each other in such a way that it is fulfilled.

When interacting with other objects an approach where the position of the vertices is corrected to fulfill the length constraint can lead to problems. The new position of the vertices can conflict with other constraints, such as non-penetration constraints. This is undesirable because often other constraints are difficult to correctly reinstate once they have been violated. Also it is unclear how forces for contacts with other objects could be calculated. To the knowledge of the author, there is no position correction that ensures no conflict with other constraints and still enforces the length constraint if the thread is known.

A force based approach pushed the system towards a configuration where the length constraint is satisfied. But it does not enforce it exactly. When the length constraints conflicts with other constraints (such as non-penetration) the thread pushes the system in the correct direction without the need to violate any other constraints. The problem with force based approaches are instabilities due to stiff forces, as described in the section above.

Implicit integration has been used to make a variety of stiff systems stable (e. g. [10] and [30]). The idea is to use the force at the next time step instead of the current. So $\vec{F}(\vec{x}(t + \Delta t)$ is used instead of $\vec{F}(\vec{x}(t))$. The term "implicit" – in contrast to "explicit" – underlines the fact, that the force at the next time step is not known in advance (because it depends on $\vec{x}(t + \Delta t)$). Instead it becomes a variable in the calculation. Here a short example will be given before jumping into the mathematical details of implicit integration.

Consider the example of the one dimensional spring introduced above and illustrated in figures 3.3 and 3.4. When the force accelerating the mass point is the force at the final position, it can only accelerate the mass point towards the springs rest position if it is not crossed by the mass point. If, on the other hand, the mass point crosses the rest point, the force is directed opposite to its velocity, thereby reducing the energy of the system. This is of course only an illustrative example, for a more general proof that implicit integration is stable consult appendix A.

While implicit integration can deal with extreme spring constants, it requires solving a square matrix of size N (N being the number of involved vertices). The thread should have vertices making such a matrix solution unfeasible. For implicit integration it is needed to solve this matrix because it sets forces acting on vertices depending on the next position of interacting vertices. In the case of the springs in the thread, only neighboring vertices are interacting. As will be shown this leads to a banded matrix solvable in linear time.

For the integration an explicit integration scheme – which will be transformed to an implicit one – with an update rule of the positions, which yields the following form has to be given:

$$\exists \boldsymbol{a}(t) \in \mathbb{R}^F \text{ and } B(t) \in \mathbb{R}^F \times \mathbb{R}^F \text{ where } B_{i,j}(t) = 0 \text{ for } i \neq j \text{ such that}$$

$$\Delta \boldsymbol{x}(t) := \boldsymbol{x}(t + \Delta t) - \boldsymbol{x}(t) = \boldsymbol{a}(t) + B(t) \cdot \boldsymbol{F}(\boldsymbol{x}(t)) \tag{3.28}$$

Where Δt is the time step. $\boldsymbol{x}$ represents the degrees of freedom in the system and F its number of dimensions. An example would be the position of vertices in 3D space (with 3 degrees of freedom for every vertex). $\boldsymbol{x}$ will sometimes be a spatial variable but it can also be a variable for other degrees of freedom, or a mixture of both. $B(t)$ must be a diagonal matrix. Interaction between the degrees of freedom are modeled through $\boldsymbol{F}$. When all degrees of freedom are treated equally, $B(t)$ degenerates to a scalar. Most integration procedures fulfill this requirement.

To give an example, the Euler-Cromer method sets $\boldsymbol{a}(t)$ and $B(t)$ to

$$\boldsymbol{a}(t) = \begin{pmatrix} \vec{v}_1(t) \\ \vec{v}_2(t) \\ \vdots \end{pmatrix} \Delta t \qquad\qquad B(t) = \begin{pmatrix} 1/m_1 \\ 1/m_2 \\ \vdots \end{pmatrix} \Delta t^2.$$

Again, for reasons of readability the parameter t will be omitted where the meaning is clear, that is, without ambiguity.

To transform (3.28) to an implicit integration scheme, $\boldsymbol{F}(\boldsymbol{x}(t))$ is replaced with $\boldsymbol{F}(\boldsymbol{x}(t+\Delta t))$:

$$\Delta\boldsymbol{x}(t) = \boldsymbol{a}(t) + B(t)\cdot\boldsymbol{F}(\boldsymbol{x}(t+\Delta t)) \tag{3.29}$$

Since this position is the outcome of the calculation, it is not known in advance. The trick is to approximate the forces by a Taylor expansion around $\boldsymbol{x}(t)$ yielding an approximation for $\boldsymbol{F}(\boldsymbol{x}(t+\Delta t)$ in dependence of $\Delta\boldsymbol{x}(t)$:

$$\boldsymbol{F}(\boldsymbol{x}(t+\Delta t)) \approx \boldsymbol{F}(\boldsymbol{x}(t)) + \sum_{j=1}^{N} \frac{\partial \boldsymbol{F}_i(\boldsymbol{x}(t))}{\partial \boldsymbol{x}_j(t)} \Delta\boldsymbol{x}_j(t)$$

Putting this back into (3.29) the substitution yields the equation that needs to be solved by the implicit integration.

$$\begin{aligned} \Delta\boldsymbol{x}(t) &= \boldsymbol{a}(t) + B(t)\left(\boldsymbol{F}(\boldsymbol{x}(t)) + \frac{\partial\boldsymbol{F}(\boldsymbol{x}(t))}{\partial\boldsymbol{x}(t)}\cdot\Delta\boldsymbol{x}(t)\right) \\ \Rightarrow \quad \left(\mathbb{1} - B(t)\frac{\partial\boldsymbol{F}(\boldsymbol{x}(t))}{\partial\boldsymbol{x}(t)}\right)\Delta\boldsymbol{x}(t) &= \boldsymbol{a}(t) + B(t)\cdot\boldsymbol{F}(\boldsymbol{x}(t)) \end{aligned} \tag{3.30}$$

If $\frac{\partial\boldsymbol{F}(\vec{x}(t))}{\partial\boldsymbol{x}(t)}$ takes any general form, then solving a linear system of size N is required, which is not feasible for a real time simulation. But for the thread there is an observation that reduces the computational burden significantly. It turns out that the forces on the degrees of freedom of a vertex only depend on the degrees of freedom of the neighboring vertices and on their neighbors.

As for every vertex there is an $\boldsymbol{x}, \boldsymbol{F}, \boldsymbol{a}$ and B indexes are needed to delineate each vertex. This is noted adding an index i the i-th vertex (for example $\boldsymbol{x})_i$, $\boldsymbol{F}_i$). Now (3.30) can be rewritten to

$$D_i\Delta\boldsymbol{x}_i + O_i^{+}\Delta\boldsymbol{x}_{i+1} + O_i^{++}\Delta\boldsymbol{x}_{i+2} + O_i^{-}\Delta\boldsymbol{x}_{i-1} + O_i^{--}\Delta\boldsymbol{x}_{i-2} = \boldsymbol{a}_i + B_i\boldsymbol{F}_i \tag{3.31}$$

where

$$D_i := \mathbb{1} - B_i \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{x}_i} \tag{3.32}$$

$$O_i^+ := -B_i \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{x}_{i+1}} \qquad O_i^{++} := -B_i \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{x}_{i+2}} \tag{3.33}$$

$$O_i^- := -B_i \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{x}_{i-1}} \qquad O_i^{--} := -B_i \frac{\partial \boldsymbol{F}_i}{\partial \boldsymbol{x}_{i-2}}. \tag{3.34}$$

$$\tag{3.35}$$

The thought behind this notation is the following: The + in O_i^+ indicates that it is the relation to the next vertex. The ++ indicates the relation to the vertex following the next vertex. Analogous to the above the $-$ in O_i^- indicates the relation to the previous vertex just as $--$ indicates the relation to the vertex previous to the previous vertex.

(3.32) - (3.34) can be expressed as:

$$M \Delta \boldsymbol{x} = \boldsymbol{a} + B \boldsymbol{F} \tag{3.36}$$

where

$$M := \begin{pmatrix} D_1 & O_1^+ & 0 & 0 & \cdots \\ O_2^- & D_2 & O_2^+ & 0 & \cdots \\ 0 & O_3^- & D_3 & O_3^+ & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix} \qquad B := \begin{pmatrix} B_1 & 0 & \cdots \\ 0 & B_2 & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

The matrix M is a banded matrix. If every vertex has n degrees of freedom the matrix has $3n - 1$ bands. This allows the system for being solved in linear time using LU decomposition [114].

3.3. Internal Forces

For the physical simulation the internal forces acting on the thread must be calculated. The integration scheme requires knowing the forces and their derivatives with respect to all degrees of freedom. The first step will be to determine the relevant degrees of freedom.

The different types of forces modeled here have already been presented in the choice of energies done in section 3.1.1. These are stretching, bending and torsion forces. Given an energy E, the force on a degree of freedom x is given by

$$F_x := -\frac{\partial E}{\partial x}$$

When calculating the derivatives of these forces, there are some caveats one has to take care of. The problematic situations are illustrated in section 3.3.3.

3.3.1. Preliminary Calculations

Some calculations are often needed so abbreviations will be introduced here as the forces are derived. These abbreviations are summarized below.

The tensor product of a vector with itself is needed quite often. A shorter notation, defined by

$$T\left(\vec{x}\right) := \vec{x} \cdot \vec{x}^T$$

will be used.

The derivative of the tangent norms occur on numerous places:

$$\frac{\partial|\vec{s}_i|}{\partial\vec{p}_i} = -\vec{t}_i$$
$$\frac{\partial|\vec{s}_i|}{\partial\vec{p}_{i+1}} = \vec{t}_i$$

And so do the derivatives of the tangents them self:

$$\begin{aligned}\frac{\partial\vec{t}_i}{\partial\vec{p}_i} &= \frac{-\mathbb{1}|\vec{s}_i| + \vec{s}_i\vec{t}_i^T}{|\vec{s}_i|^2} = \frac{T\left(\vec{t}_i\right) - \mathbb{1}}{|\vec{s}_i|} \\ \frac{\partial\vec{t}_i}{\partial\vec{p}_{i+1}} &= \frac{\mathbb{1}|\vec{s}_i| - \vec{s}_i\vec{t}_i^T}{|\vec{s}_i|^2} = \frac{\mathbb{1} - T\left(\vec{t}_i\right)}{|\vec{s}_i|}\end{aligned}$$

The derivative of the cross product with respect to one of its components is:

$$\frac{\partial}{\partial\vec{x}}\left(\vec{x} \times \vec{y}\right) = \begin{pmatrix} 0 & y_3 & -y_2 \\ -y_3 & 0 & y_1 \\ y_2 & -y_1 & 0 \end{pmatrix} =: [\vec{y}]$$

For a given $\vec{y}$, $[\vec{y}]$ has the property, that for any vector $\vec{z}$:

$$[\vec{y}] \cdot \vec{z} = \vec{z} \times \vec{y}$$

3.3.2. Degrees of Freedom

The positions $\vec{p}_i, 1 \leq i \leq N$ of the vertices describing the discretized center line of the thread are degrees of freedom that have to be integrated.

Another degree of freedom is needed to model the twist of the thread. One possible choice would be to use would be the angle between the Bishop frame and the material frame. In the following this will be called the the twist angle. But this choice is not elegant: This degree of freedom is associated with segments as opposed to vertices. This causes an asymmetry. It cannot be paired one by one with the positional degrees of freedom because it is not associated with vertices. Angles can be defined for the vertices by averaging the angles of adjacent segments. Also, as stated before, the angle is arbitrary under a constant offset. This is not a problem per se, but nice if it can be avoided.

By taking the difference of the twist angle of adjacent segments which is associated with every vertex, the author of this thesis feels this is a more natural way of determining the degree of freedom describing the twisting behavior of the thread. The delta angle has already been introduced in (3.23) and is denoted by $\delta\Phi_i$.

With this notation the inelegance described above is avoided. However, this introduces another situation one has to be aware of. The positional degrees of freedom are independent of each other, which means

$$\frac{\partial\vec{p}_i}{\partial\vec{p}_j} = 0 \qquad \text{for } i \neq j.$$

But the $\delta\Phi_i$ are not. The $\Phi(s_i)$ in (3.23) are independent, allowing to find the dependence of the $\delta\Phi_i$.

$$\frac{\partial(\delta\Phi_i)}{\partial(\delta\Phi_j)} = \begin{cases} 1 & \text{if } i = j \\ -1 & \text{if } i = j-1 \text{ or } j = i-1 \\ 0 & \text{otherwise} \end{cases} \tag{3.37}$$

As discussed later, the twist can also be excluded from the implicit integration. For this the “quasi static updated” from Bergou et al. [13] is done prior to an integration step.

The forces are found by deriving the discrete energies with respect to the degrees of freedom. Most of the energies are independent of the twist. In these cases only the positions of the vertices are relevant and the forces will be derived as if these would be the only degrees of freedom.

3.3.3. Problems when calculating the derivatives of the forces

In 3.30 the force is approximated in the neighborhood of $\boldsymbol{x}(t)$ using a first order Taylor expansion. It is assumed that this approximates the force at the next position $\boldsymbol{x}(t+\Delta t)$ well. The values needed for the integration in (3.32)-(3.34) are found with the help of the derivatives of $\boldsymbol{F}$. It is important to remember, that the goal of the Taylor expansion is not to improve accuracy but stability and thereby visual plausibility. It assumes that a linear approximation of the force describes it accurately in the neighborhood of $\boldsymbol{x}(t)$. A first order approximation of the force leads to a second order approximation of the related energy potential.

But certain types of forces lead to badly formed first order approximation causing unstable or incorrect behavior of the system. For very small steps, the first order Taylor expansion is always good, but when the system makes wider steps the approximation can be completely wrong. Often the incorrect approximation for wider steps is the cause of the system making this big step in the first place.

When finding the values for (3.32)-(3.34), $\frac{\partial \boldsymbol{F}}{\partial \boldsymbol{x}}$ is good most of the times. But when it is not, it must be modified to describe the force around $\boldsymbol{x}(t)$ better. Here two problematic cases that occur during derivation of the forces on the thread are described.

Problems with $1/|\vec{s}_i|$

When deriving the bending forces in section 3.3.5, terms like $\vec{t}_{i+1}/|\vec{s}_i|$ occur (remember the definition of $\vec{s}_i$ and $\vec{t}_{i+1}$ from (3.19) and (3.20)). This means the force along $\vec{t}_{i+1}$ is reduced, when the i-th segment is stretched.

In Figure 3.5 the effect on the force along $\vec{t}_{i+1}$ in dependency of $|\vec{s}_i|$ is shown. Also the linear approximation is displayed.

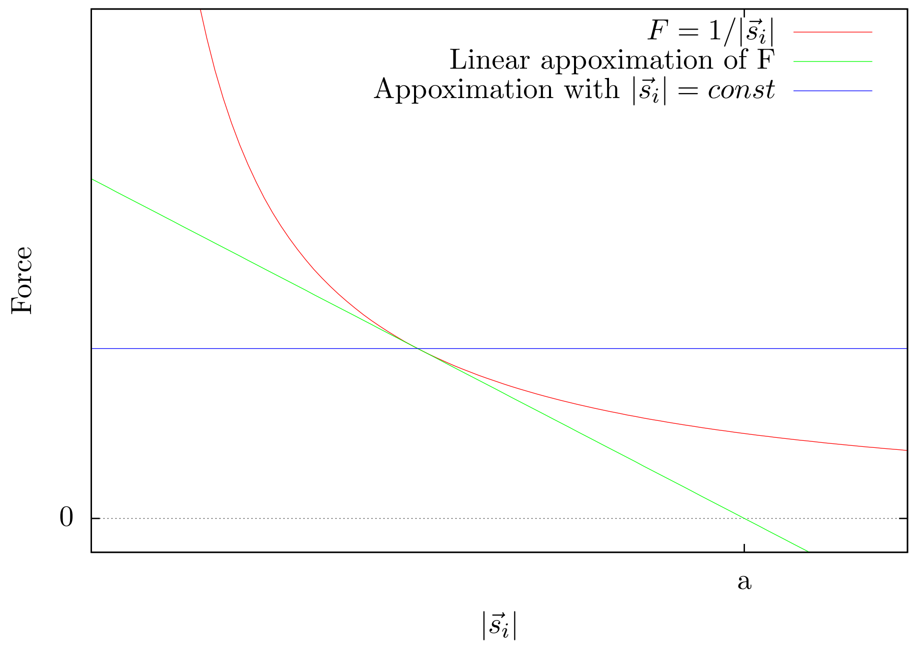


Figure 3.5.: Linear approximation of $1/|\vec{s}_i|$.

This approximation makes the algorithm assume, that at $|\vec{s}_i| = a$ (see figure 3.5) the effect on the force in direction $\vec{t}_i$ vanishes, and worse, for $|\vec{s}_i| > a$ is even negative.

This is completely wrong. This is the primary problem, namely, the need to believe that the simulation is describing plausible physical behavior.

Another linear approximation for the force must be found. A simple solution is to assume that $|\vec{s}_i|$ does not change during the time step making it a constant. This simple adjustment leads to an approximation that makes the system stable and plausible again.

Figure 3.5 also shows this new approximation of $\vec{F}$. It is qualitatively better – meaning that it leads to better results – has already been explained. Figure 3.5 illustrates that it is quantitatively a good fit, meaning how close it is to the correct solution.

Problems with local maxima

Imagine an energy potential having a local maximum at $x(t)$ as illustrated for a one dimensional case with one mass point which position is described by x in figure 3.6.

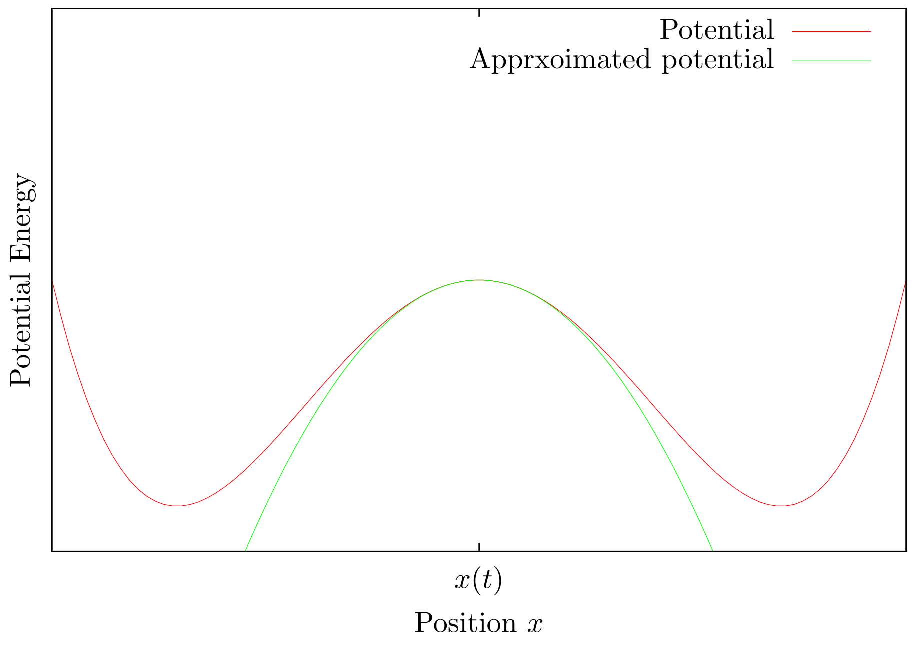


Figure 3.6.: Approximation of potential with local maximum.

Approximating the force linearly results in a quadratic approximation of the potential, as also shown in figure 3.6. This should immediately give one a bad feeling, because this quadratic potential results in a repulsing force growing with distance. If ones examines (3.29) for the one dimensional case (for simplicity here $a(t) = 0$ is

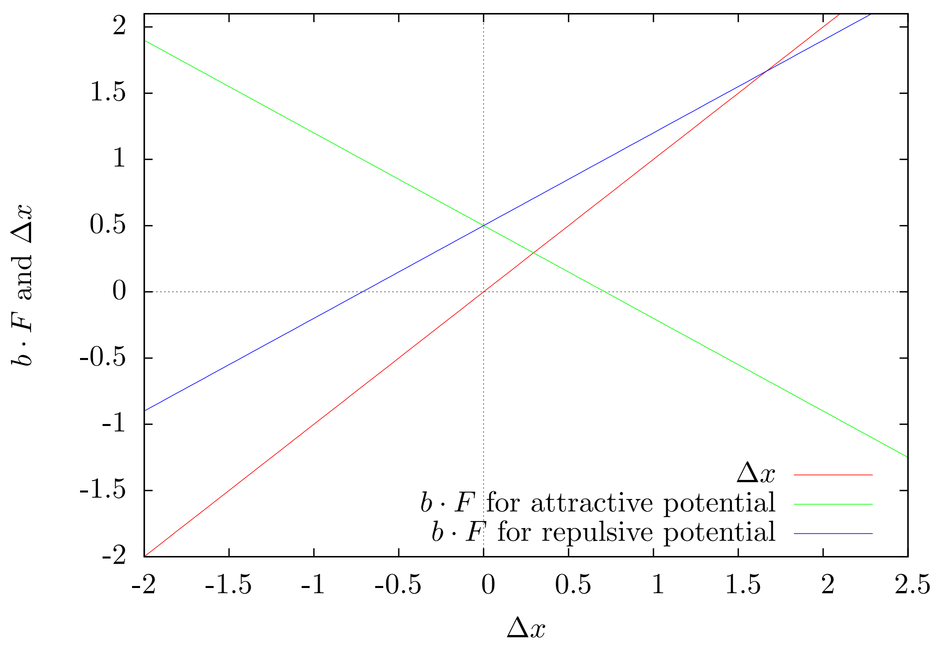


Figure 3.7.: Graphical solution of (3.29) for attractive and repulsive potentials. The solution of the equation is where $b \cdot F$ intersects with Δx

assumed) one finds that the equation can be solved graphically. Both sides of (3.29) are plotted in dependency of Δx. The left side simply becomes the identity while the right side becomes a straight line with slope $B(t) \cdot \frac{\partial F}{\partial x}$. $B(t)$ is assumed to be positive. The solution of (3.29) is, where the two lines intersect.

The situation is illustrated for an attractive and a repulsive local approximation of the energy potential in figure 3.7. If it is attractive (meaning the parabola is open to the top), the slope of the right hand side term is negative and the intersection is close to $\vec{x}(t)$. But in case of the repulsive potential, the slope is positive and the intersection can be very far from $\Delta x = 0$ causing the system to jump. Looking back at 3.6, at a position far from $\Delta x = 0$, the potential is again strongly attractive towards the original position x and the quadratic potential is too. So the mass point is moved back towards it's original position where the process starts again. This process describes an oscillating behavior which is undesirable.

To avoid this problem it should always be guaranteed that the approximation of the energy potential is attractive.

3.3.4. Stretching Forces

The stretching forces are easily found by deriving (3.21) with respect to the degrees of freedom. Since it is independent of the twist, only the derivative with respect to the positions are needed. To find the forces first the derivatives of the individual summands are deduced:

$$\begin{aligned}
\vec{S}_i^- &:= \frac{\partial}{\partial \vec{p}_i}\left(\frac{l_{i-1}}{l}\left(\frac{|\vec{s}_{i-1}|}{l_{i-1}}-1\right)^2\right) = \frac{2}{l}\left(\frac{|\vec{s}_{i-1}|}{l_{i-1}}-1\right)\vec{t}_{i-1}\\
\vec{S}_i^+ &:= \frac{\partial}{\partial \vec{p}_i}\left(\frac{l_i}{l}\left(\frac{|\vec{s}_i|}{l_i}-1\right)^2\right) = -\frac{2}{l}\left(\frac{|\vec{s}_i|}{l_i}-1\right)\vec{t}_i
\end{aligned}$$

The stretching forces are than easily found:

$$\begin{aligned}
\vec{F}_1^S &= -\frac{k_S}{2}\vec{S}_1^+\\
\vec{F}_N^S &= -\frac{k_S}{2}\vec{S}_N^-\\
\vec{F}_i^S &= -\frac{k_S}{2}\left(\vec{S}_i^+ + \vec{S}_i^-\right) && \text{for } 1 < i < N
\end{aligned}$$

The derivatives of these forces are found by deriving $\vec{S}_i^-$ and $\vec{S}_i^+$. Here only the non-vanishing terms are listed.

$$\begin{aligned}
\frac{\partial \vec{S}_i^-}{\partial \vec{p}_{i-1}} &= \frac{2}{l}\left[\frac{-\vec{t}_{i-1}}{l_{i-1}}\cdot \vec{t}_{i-1}^{\,T} + \left(\frac{|\vec{s}_{i-1}|}{l_{i-1}}-1\right)\cdot\frac{T\left(\vec{t}_{i-1}\right)-\mathbb{1}}{|\vec{s}_{i-1}|}\right]\\
&= -\frac{2}{l\cdot l_{i-1}}\left[\mathbb{1}+u_{i-1}\left(T\left(\vec{t}_{i-1}\right)-\mathbb{1}\right)\right] && (3.38)\\
\frac{\partial \vec{S}_i^-}{\partial \vec{p}_i} &= \frac{2}{l\cdot l_{i-1}}\left[\mathbb{1}+u_{i-1}\left(T\left(\vec{t}_{i-1}\right)-\mathbb{1}\right)\right] = -\frac{\partial \vec{S}_i^-}{\partial \vec{p}_{i-1}} && (3.39)\\
\frac{\partial \vec{S}_i^+}{\partial \vec{p}_i} &= \frac{2}{l\cdot l_i}\left[\mathbb{1}+u_i\left(T\left(\vec{t}_i\right)-\mathbb{1}\right)\right] && (3.40)\\
\frac{\partial \vec{S}_i^+}{\partial \vec{p}_{i+1}} &= -\frac{2}{l\cdot l_i}\left[\mathbb{1}+u_i\left(T\left(\vec{t}_i\right)-\mathbb{1}\right)\right] = -\frac{\partial \vec{S}_i^+}{\partial \vec{p}_{i+1}} && (3.41)
\end{aligned}$$

with $u_i := l_i/|\vec{s}_i|$. (3.38)-(3.41) allows one to write the force derivatives as:

$$\begin{aligned}
\frac{\partial \vec{F}_1^S}{\partial \vec{p}_1} &= -\frac{k_S}{2}\frac{\partial S_1^+}{\partial \vec{p}_1} & \frac{\partial \vec{F}_1^S}{\partial \vec{p}_2} &= -\frac{k_S}{2}\frac{\partial S_1^+}{\partial \vec{p}_2} \\
\frac{\partial \vec{F}_N^S}{\partial \vec{p}_N} &= -\frac{k_S}{2}\frac{\partial S_N^-}{\partial \vec{p}_N} & \frac{\partial \vec{F}_N^S}{\partial \vec{p}_{N-1}} &= -\frac{k_S}{2}\frac{\partial S_N^-}{\partial \vec{p}_{N-1}} \\
\frac{\partial \vec{F}_i^S}{\partial \vec{p}_{i-1}} &= -\frac{k_S}{2}\frac{\partial S_i^-}{\partial \vec{p}_{i-1}} & && 1 < i < N \\
\frac{\partial \vec{F}_i^S}{\partial \vec{p}_i} &= -\frac{k_S}{2}\left(\frac{\partial S_i^-}{\partial \vec{p}_i} + \frac{\partial S_i^+}{\partial \vec{p}_i}\right) & && 1 < i < N \\
\frac{\partial \vec{F}_i^S}{\partial \vec{p}_{i+1}} &= -\frac{k_S}{2}\frac{\partial S_i^+}{\partial \vec{p}_{i+1}} & && 1 < i < N
\end{aligned}$$

Again, only the non vanishing terms are shown.

Ensuring Stability: The reason the u_i has been extracted in (3.38)-(3.41) is that there is a special case causing instability which can be avoided by redefining u_i. To illustrate the problem let $\vec{v}$ be a normalized vector perpendicular to $\vec{t}_i$. Then

$$\frac{\partial S_i^+}{\partial \vec{p}_i}\vec{v} = \frac{2}{l \cdot l_i}\left[1 + u_i\left(T\left(\vec{t}_i\right) - \mathbb{1}\right)\right]\vec{v} = \frac{2}{l \cdot l_i}\left[\vec{v} + u_i\left(-\vec{v}\right)\right] = \frac{2}{l \cdot l_i}(1 - u_i)\vec{v}$$

This means that $\vec{v}$ is an eigenvector of $\frac{\partial S_i^+}{\partial \vec{p}_i}$ whose eigenvalue is negative if $u_i > 1$. A negative eigenvalue of $\frac{\partial S_i^+}{\partial \vec{p}_i}$ causes $\frac{\partial \vec{F}_i^S}{\partial \vec{p}_i}$ to have a positive eigenvalue which means that the corresponding potential is repulsive in the direction of the corresponding eigenvector. This causes the instability problems described in section 3.3.3. The problem can be avoided by ensuring that u_i is always smaller than or equal to 1, which has been guaranteed by redefining u_i to

$$u_i := \min\left(\frac{l_i}{|\vec{s}_i|}, 1\right).$$

u_i then only deviates from it's physical value, when the thread is compressed which rarely happens during the simulation.

3.3.5. Bending forces

The discretized bending energy in (3.22) involves the angle Θ_i between adjacent segments. Assuming this angle is small, it is approximated by the distance between the tangents of the corresponding segments (see figure 3.3.5):

$$\Theta_i \approx \left|\vec{t}_i - \vec{t}_{i-1}\right| \qquad 1 < i < N$$

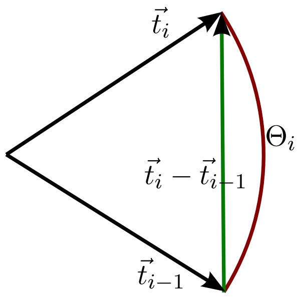


Figure 3.8.: Approximating Θ_i with $\vec{t}_i - \vec{t}_{i-1}$.

Note that Θ_i is not defined for the outermost vertices since no bending can be defined there.

Remembering the meaning and relation of the values on segments adjacent to Θ_i and those belonging to Θ_i can be confusing. For guidance, figure 3.9 illustrates all quantities involved.

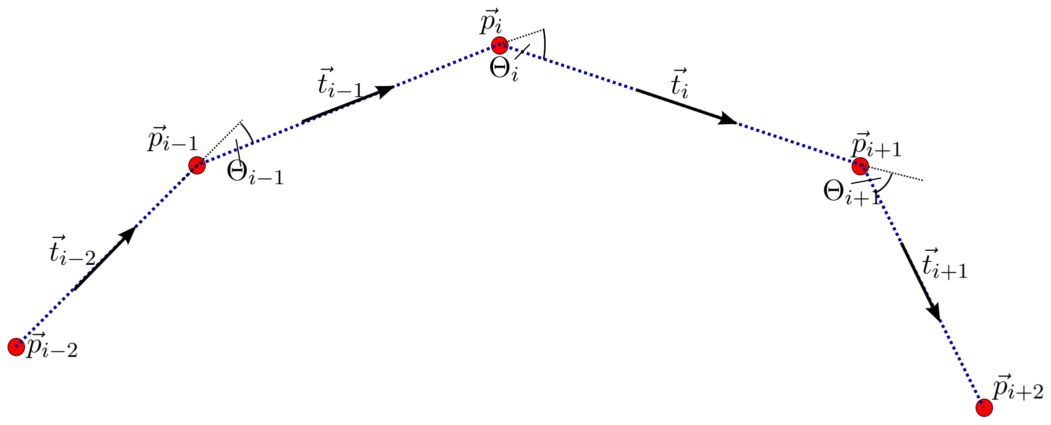


Figure 3.9.: Important quantities when deriving the bending energy.

Similar to the derivation of the stretching forces, first the individual terms in (3.22) – which are the squared values of Θ_i – are derived. For Θ_{i+1} the calculation is:

$$A_i^+ := \frac{1}{2}\frac{\partial \Theta_{i+1}^2}{\partial \vec{p}_i} = \left(\frac{\mathbb{1} - T\left(\vec{t}_i\right)}{|\vec{s}_i|}\right)\left(\vec{t}_{i+1} - \vec{t}_i\right) \approx \frac{\vec{t}_{i+1} - \vec{t}_i}{|\vec{s}_i|} \qquad 1 \leq i < N-1$$

$$A_i^+ := \frac{\vec{t}_{i+1} - \vec{t}_i}{\left|\vec{s}_i\right|} \qquad\qquad 1 \leq i < N-1 \tag{3.42}$$

The last approximation is again based on the assumption, that $\vec{t}_i$ and $\vec{t}_{i+1}$ are similar, which implies $\vec{t}_i \cdot \vec{t}_{i+1} \approx 1$. This means

$$\left(\mathbb{1} - T\left(\vec{t}_i\right)\right)\left(\vec{t}_{i+1} - \vec{t}_i\right) = \left(\vec{t}_{i+1} - \vec{t}_i\right) - \left(\left(\vec{t}_i \cdot \vec{t}_{i+1}\right)\vec{t}_i - \vec{t}_i\right), \approx \left(\vec{t}_{i+1} - \vec{t}_i\right)$$

justifying the approximation in (3.42).

Analogously one gets:

$$A_i^- := \frac{1}{2}\frac{\partial \Theta_{i-1}^2}{\partial \vec{p}_i} = \frac{\vec{t}_{i-1} - \vec{t}_{i-2}}{\left|\vec{s}_{i-1}\right|} \qquad\qquad 2 < i \leq N \tag{3.43}$$

$$\begin{aligned} A_i &:= \frac{1}{2}\frac{\partial \Theta_i^2}{\partial \vec{p}_i} = \left(\vec{t}_{i-1} - \vec{t}_i\right) \cdot \left(\frac{1}{\left|\vec{s}_i\right|} + \frac{1}{\left|\vec{s}_{i-1}\right|}\right) \\ &= -\left(A_{i-1}^+ + A_{i+1}^-\right) \qquad\qquad 1 < i < N \end{aligned} \tag{3.44}$$

For the bounds be aware that Θ_1 and Θ_N are not defined.

The bending forces are defined in terms of A_i^-, A_i and A_i^+. The general form is:

$$\vec{F}_i^B := -\frac{\partial E^B}{\partial \vec{p}_i} = -\frac{\partial}{\partial \vec{p}_i}\sum_{j=2}^{N-1}\Theta_j^2.$$

This yields:

$$\begin{aligned} &\vec{F}_1^B = -k_B A_1^+ && \vec{F}_2^B = -k_B\left(A_2 + A_2^+\right) \\ &\vec{F}_N^B = -k_B A_N^- && \vec{F}_{N-1}^B = -k_B\left(A_{N-1} + A_{N-1}^-\right) \\ &\vec{F}_i^B = -k_B\left(A_i^- + A_i + A_i^+\right) && \qquad 2 < i < N-1 \end{aligned} \tag{3.45}$$

For the integration scheme, the derivatives of this force with respect to $\vec{p}_j$ are needed.

$$\frac{\partial A_i^-}{\partial \vec{p}_{i-2}} = \frac{\mathbb{1} - T\left(\vec{t}_{i-2}\right)}{\left|\vec{s}_{i-1}\right| \left|\vec{s}_{i-2}\right|} \qquad 2 < i \leq N \tag{3.46}$$

$$\frac{\partial A_i^-}{\partial \vec{p}_{i-1}} = \frac{1}{\left|\vec{s}_{i-1}\right|} \left(\frac{T\left(\vec{t}_{i-1}\right) - \mathbb{1}}{\left|\vec{s}_{i-1}\right|} + \frac{T\left(\vec{t}_{i-2}\right) - \mathbb{1}}{\left|\vec{s}_{i-2}\right|} \right) \qquad 2 < i \leq N \tag{3.47}$$

$$\frac{\partial A_i^-}{\partial \vec{p}_{i}} = \frac{\mathbb{1} - T\left(\vec{t}_{i-1}\right)}{\left|\vec{s}_{i-1}\right| \left|\vec{s}_{i-1}\right|} \qquad 2 < i \leq N \tag{3.48}$$

$$\frac{\partial A_i^+}{\partial \vec{p}_{i+2}} = \frac{\mathbb{1} - T\left(\vec{t}_{i+1}\right)}{\left|\vec{s}_{i}\right| \left|\vec{s}_{i+1}\right|} \qquad 1 < i < N \tag{3.49}$$

$$\frac{\partial A_i^+}{\partial \vec{p}_{i+1}} = \frac{1}{\left|\vec{s}_{i}\right|} \left(\frac{T\left(\vec{t}_{i+1}\right) - \mathbb{1}}{\left|\vec{s}_{i+1}\right|} + \frac{T\left(\vec{t}_{i}\right) - \mathbb{1}}{\left|\vec{s}_{i}\right|} \right) \qquad 1 < i < N \tag{3.50}$$

$$\frac{\partial A_i^+}{\partial \vec{p}_{i}} = \frac{\mathbb{1} - T\left(\vec{t}_{i}\right)}{\left|\vec{s}_{i}\right| \left|\vec{s}_{i}\right|} \qquad 1 < i < N \tag{3.51}$$

The remaining terms for A_i^- and A_i^+ vanish. The terms for A_i follow from (3.44). The derivatives of the bending forces are now found by inserting (3.46) - (3.51) into (3.45).

3.3.6. Torsion forces

So far the derivatives of the energies with respect to $\delta\Phi_i$ have been omitted. This is justified by the fact that all but the twisting energy are functions of only $\vec{p}_j$, which are not seen as functions of $\delta\Phi_i$. In other words:

$$\frac{\partial \vec{p}_j}{\partial \delta\Phi_i} = 0$$

Changing $\delta\Phi_i$ changes the material frame, but only by rotating it around $\vec{t}_j$ which does not affect any of the vertex positions.

The discretized energy for the twisting forces on the other hand is a function of $\delta\Phi_i$. While the derivatives between the $\delta\Phi_i$ are given in (3.37), the derivatives with respect to the vertex positions are a bit more complicated. How does $\delta\Phi_i$ change when $\vec{p}_j$ changes? The question involves asking how the Bishop Frame moves with $\vec{p}_j$.

Bishop frame and parallel transport

To define the twisting energy for the discrete case, the notion of parallel transport is needed. The idea and formalism of using parallel transport for the twisting energy is taken from Bergou et al. [13].

Recall the Bishop frame section 3.1.1. It was defined by having vanishing torsional energy, which reduces the Darboux vector to

$$\vec{\Omega}_B(s) := -\omega_2(s)\vec{M}_1(s) + \omega_1(s)\vec{M}_2(s).$$

A material frame is assigned to every segment. For the smooth case the twisting energy depends on the angle between the Bishop frame and the material frame. This angle is also needed in the discretized case. Parallel transport will play an important role in finding it. The piece-wise linear discretized center line is smoothed at it's kinks (the vertices) by a circle segment which continuously merges into the segments on each side of the kink. The radius of the circle segment is not important because it will turn out that the result is independent of it. The only condition on the radius is that adjacent circle segments do not overlap. This can be guaranteed by ensuring that circle segments do not intersect with the midpoints of the connected segments (see figure 3.10).

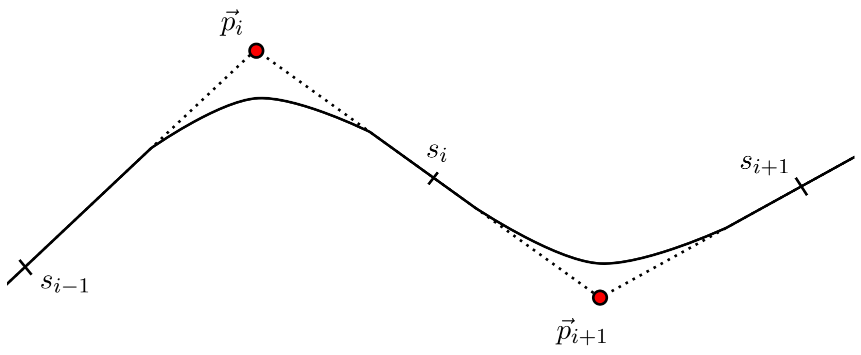


Figure 3.10.: The transition between adjacent segments is smoothed with circle segments.

First the transportation of the *Bishop Frame* will be analyzed. For the straight part of the segments, the Darboux Vector vanishes and the Bishop frame is constant. Let $\vec{v}$ be a vector in the Bishop Frame on the i-th segment. Let further $\{s_i | 1 \leq i < N, s_i \in [0,1]\}$ denote the parameter s at the midpoints of the segments. Note that $\vec{t}_i = \vec{T}(s_i)$ and $\vec{t}_{i+1} = \vec{T}(s_{i+1})$.

The parallel transport operator P_i transforms $\vec{v}$ into the Bishop frame of the $(i+1)$-th Segment (see figure 3.11). It is defined in such a way that $P_i(\vec{v})$ expressed in the Bishop frame of the $(i+1)$-th segment is the same as $\vec{v}$ expressed in the Bishop frame of the i-th segment.

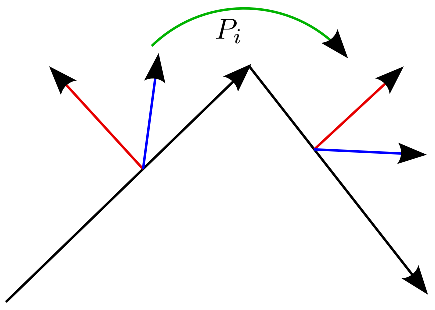


Figure 3.11.: Parallel transport of the material frame form one segment to the next. The black arrows denote the tangents $\vec{t_i}$ and $\vec{t}_{i+1}$. The red and blue arrows denote the remaining elements of the Bishop frame $\vec{M}_1^0$ and $\vec{M}_2^0$ respectively. The green arrow illustrates how the parallel transport P_i transforms one Bishop frame into the next.

To find the transportation from the i-th segment to $i+1$ the segment $\vec{v}$ is parametrized with s such that it is always $\vec{v}$ transformed to the material frame at s.

$$\vec{v}(s) := v_1\vec{T}^0(s) + v_2\vec{M}_1^0(s)v_3\vec{M}_2^0(s)$$

Equation (3.7)-(3.9) define a differential equation for $\vec{s}$:

$$\begin{aligned}\frac{\partial\vec{v}(s)}{\partial s} &= v_1\frac{\partial\vec{T}^0(s)}{\partial s} + v_2\frac{\partial\vec{M}_1^0(s)}{\partial s}v_3\frac{\partial\vec{M}_2^0(s)}{\partial s}\\ &= v_1\left(\Omega_B(s)\times\vec{T}^0(s)\right) + v_2\left(\Omega_B(s)\times\vec{M}_1^0(s)\right) + v_3\left(\Omega_B(s)\times\vec{M}_2^0(s)\right)\\ &= \Omega_B(s)\times\left(v_1\vec{T}^0(s) + v_2\vec{M}_1^0(s) + \vec{M}_2^0(s)\right) = \Omega_B(s)\times\vec{v}(s)\end{aligned}$$

So $P_i(\vec{v})$ is obtained by integrating

$$\frac{\partial\vec{v}(s)}{\partial s} = \Omega_B(s)\times\vec{v}(s) \tag{3.52}$$

from s_i to s_{i+1} with starting value $\vec{v}(s_i) = \vec{v}$ and setting $P(\vec{v}) := \vec{v}(s_{i+1})$. The fact that $\vec{\Omega}_B$ has no component tangential to $\vec{T}(s)$ combined with (3.7) implies, that $\vec{\Omega}_B(s)$ is parallel to $\vec{T}(s)\times\frac{\partial\vec{T}(s)}{\partial s}$ and therefor always parallel to $\vec{t_i}\times\vec{t}_{i+1}$. Equation (3.52) means that $\vec{v}(s)$ rotates around $\Omega_B(s)$ and therefore in this case around $\vec{t_i}\times\vec{t}_{i+1}$.

So P_i is a rotation around $\vec{t}_i \times \vec{t}_{i+1}$. Since $P_i(\vec{t}_i) = \vec{t}_{i+1}$, P_i must be a rotation by the angle between the i-th and $(i+1)$-th segment. Intuitively this result makes sense, since the Bishop frame is supposed to be twist free and P_i transforms the material frame without rotating $\vec{M}_1$ and $\vec{M}_2$ around $\vec{T}$.

P_i is known as the parallel transport from $\vec{t}_i$ to $\vec{t}_{i+1}$ along a geodesic of the unit sphere. This notion will play an important role in the next section.

The derivative of $\delta\Phi_i$ with respect to $\vec{p}_j$

The goal of the following section is to derive an expression for the derivative of the twist angle with respect to the Cartesian coordinates of the vertices. This will be done utilizing parallel transport. The idea of finding the derivative of the twist angle with the help of parallel transport is taken from [13]. They use holonomy to find the derivative. Here a different geometric approach is given, but the results are the same.

Assume two adjacent segments with tangents $\vec{t}_{n-1}$ and $\vec{t}_n$. In a following time step the end vertex of the second segment moves yielding a new tangent $\vec{t}_n'$ for the segment. How much did the twist angle $\delta\Phi_n$ between the segments change by this movement?

As always, the movement of the vertex is assumed to be linear. So the tangent moves in the plane spanned by $\vec{t}_n$ and $\vec{t}_n'$. Since it is always normalized its movement is a rotation around the normal of the plane. This normal is proportional to $\vec{t}_n \times \vec{t}_n'$. The material frame is therefore transformed by this rotation around $\vec{t}_n \times \vec{t}_n'$ (see also [13]). Figure 3.12 illustrates this.

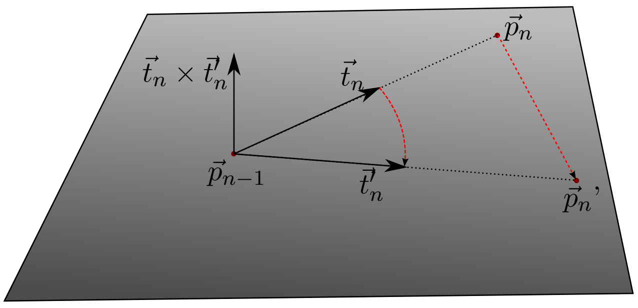


Figure 3.12.: Following the movement of $\vec{p}_n$ to $\vec{p}_n'$, $\vec{t}_n$ rotates onto $\vec{t}_n'$.

The twist angle before the movement is given by the angle between the material frame of the n-th segment and the material frame of th $(n-1)$-th segment transported onto the n-th segment via parallel transport. After the movement it is given the same way, but with the moved n-th segment (which now has the tangent $\vec{t}_n'$).

The change of the twist angle is thus given by the difference between these angles.

It can therefore be found by parallel transporting the material frame of the n-th segment from $\vec{t}_{n-1}$ onto $\vec{t}_n$. From there the change of the tangent corresponds to the parallel transport from $\vec{t}_n$ to $\vec{t}'_n$. Repeating the process, but directly from $\vec{t}_{n-1}$ to $\vec{t}'_n$ (without the extra stop on $\vec{t}_n$) and measuring the angles between the resulting frames yields the desired change of twist angle.

The same result is obtained when the material frame of the $(n-1)$-th segment is parallel transported from $\vec{t}_{n-1}$ onto $\vec{t}_n$, from there into $\vec{t}'_n$ and back onto $\vec{t}_{n-1}$. In this case the change of twist angle is the angle between the original material frame and the transported one.

Since the parallel transport always transforms the tangent of a segment into the tangent of the target segment, the angle $\Delta\delta\Phi_n$ is a rotation around $\vec{t}_{n-1}$. It is not required to parallel transform the whole material frame. Using any vector orthogonal to $\vec{t}_{n-1}$ yields the same result.

Let $P_{n-1,n}$ be the parallel transport operator from $\vec{t}_{n-1}$ to $\vec{t}_n$ and $P'_{n-1,n}$ from $\vec{t}_{n-1}$ to $\vec{t}_n$. $P_{\Delta\vec{p}_n}$ is the parallel transport for the transition from $\vec{t}_n$ to $\vec{t}'_n$. If $\vec{m}$ is a vector orthogonal to $\vec{t}_{n-1}$, the change of the twist angle $\Delta(\delta\Phi_n)$ is the angle between $\vec{m}$ and $((P'_{n-1,n})^{-1} \circ P_{\Delta x_n} \circ P_{n-1,n})(\vec{m})$.

Figure 3.13(a) and 3.13(b) should support intuition by giving two examples. One example where the change of $\vec{p}_{n+1}$ does not change the twist angle, and one where it does.

An alternate perspective on this, is the parallel transport along geodesics on the surface of a unit sphere. $\vec{t}_{n-1}, \vec{t}_n$ and $\vec{t}'_n$ build a triangle which has geodesics as sides (see figure 3.14). $\vec{m}$ is tangent to the surface of the triangle at $\vec{t}_{n-1}$. Transporting it around the triangle will result in $\vec{m}$ being rotated by the desired angle $\Delta(\delta\Phi_n)$. The case in figure 3.13(a) would correspond to a degenerated triangle, where $\vec{t}_{n-1}$, $\vec{t}_n$ and $\vec{t}'_n$ are located on the same geodesic.

From the differential geometry of a sphere, it is known that this angle is independent of the original orientation of $\vec{m}$ (as long as it is tangent to the spheres surface), which is in agreement with $\Delta(\delta\Phi_n)$ being independent of $\vec{m}$.

Let $\vec{e}_1, \vec{e}_2, \vec{e}_3$ with

$$\begin{aligned} \vec{e}_1 &:= \frac{\vec{t}_{n-1} \times \vec{t}_n}{\left|\vec{t}_{n-1} \times \vec{t}_n\right|} \\ \vec{e}_2 &:= \vec{t}_n \\ \vec{e}_3 &:= \vec{e}_1 \times \vec{e}_2 \end{aligned}$$

be an orthonormal coordinate system and $\partial_1, \partial_2, \partial_3$ the directional derivatives along the corresponding axis. Using these $\frac{\partial\delta\Phi_n}{\partial\vec{p}_{n+1}}$ can be written as:

$$\frac{\partial(\delta\Phi_n)}{\partial\vec{p}_{n+1}} = \sum_{i=1}^{3} \vec{e}_i \partial_i (\delta\Phi_n) \tag{3.53}$$

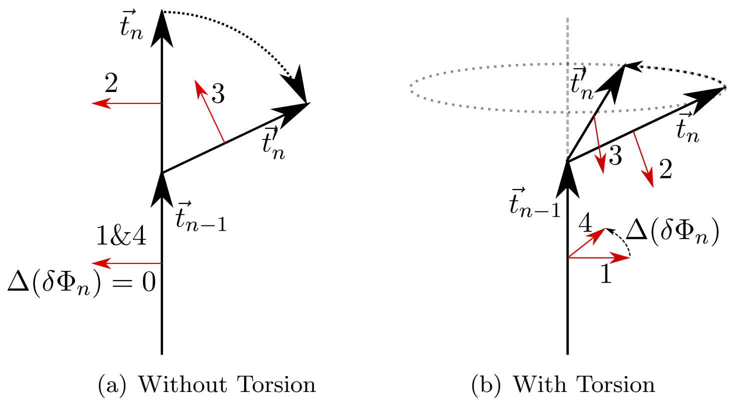


Figure 3.13.: Two cases of parallel transport to calculate the change in the twist angle. The red arrow denotes a vector parallel to the tangents that is parallel transported. The numbers denote the steps. From $\vec{t}_{n-1}$ (1) to $\vec{t}_n$ (2) to $\vec{t}'_n$ (3) and back to $\vec{t}_{n-1}$ (4).
Left: The n-th segment moves, such that $\vec{t}_n$, $\vec{t}'_n$ and $\vec{t}_{n-1}$ are in a plane. All parallel transports are rotations around the normal of this plane. When the transported vector is returned to $\vec{t}_{n-1}$, it is the same as the original. Therefore the twist angle has not changed.
Right: The n-th segment is moves outside of the plane. When the vector is parallel transported back to $\vec{t}_{n-1}$, it has an angle to the original vector. This angle is $\Delta(\delta\Phi_n)$.

When moving $\vec{p}_{n+1}$ in the $\vec{e}_2$ direction, $\vec{t}_n$ does not change. Thus $\partial_2(\delta\Phi_n)$ vanishes. A movement in $\vec{e}_3$ direction would yield a degenerated triangle (this is the case pictured in figure 3.13(a)). Therefore $\partial_3(\delta\Phi_n)$ also vanishes leaving $\partial_1(\delta\Phi_n)$ to be solved.

When $\vec{p}_{n+1}$ is moved along $\vec{e}_1$, $\vec{t}_n$ rotates around a geodesic which is perpendicular to the geodesic through $\vec{t}_n$ and $\vec{t}_{n-1}$. Figure 3.15 pictures the situation and introduces the angles α, β, a, b and c.

The position of $\vec{t}'_n$ when moving along $\vec{e}_1$ is completely described by the angle b. Therefore $\frac{\partial(\delta\Phi_n)}{\partial b}|_{b=0}$ can be factored out from $\partial_1(\delta\Phi_n)|_{b=0}$ using the chain rule:

$$\partial_1(\delta\Phi_n) = \frac{\partial(\delta\Phi_n)}{\partial b}\partial_1 b\bigg|_{b=0} = \left(\lim_{b\to 0}\frac{\Delta(\delta\Phi_n)}{b}\right)\partial_1 b|_{b=0} = \left(\lim_{b\to 0}\frac{\Delta(\delta\Phi_n)}{b}\right)\frac{1}{|\vec{s}_n|}$$

As stated above $\Delta(\delta\Phi_n)$ is the rotation of a vector when parallel transported around the triangle in 3.15. This is a quantity known in differential geometry as the angle deficit and is equal to the difference between the sum of the angles in the

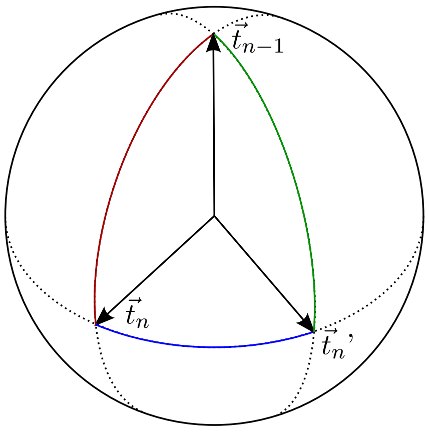


Figure 3.14.: Geodesic triangle formed by $\vec{t}_{n-1}$, $\vec{t}_n$ and $\vec{t}_n'$

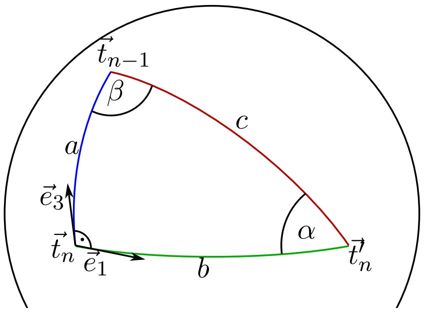


Figure 3.15.: Right triangle on the surface of a sphere, defining a, b, c, α and β. This picture corresponds to a situation where $\vec{p}_{n+1}$ is moved in $\vec{e}_1$ direction.

triangle and π.

$$\Delta(\delta\Phi_n) = \pi - \frac{\pi}{2} - \alpha - \beta = \frac{\pi}{2} - \alpha - \beta$$

$$\Rightarrow \quad \lim_{b\to 0} \frac{\Delta(\delta\Phi_n)}{b} = \lim_{b\to 0} \frac{\pi/2 - \alpha - \beta}{b} = -\lim_{b\to 0} \frac{\alpha - \alpha\mid_{b=0}}{b} - \lim_{b\to 0} \frac{\beta - \beta\mid_{b=0}}{b}$$

$$= -\left.\frac{\partial\alpha}{\partial b}\right|_{b=0} - \left.\frac{\partial\beta}{\partial b}\right|_{b=0}$$

Note that $\alpha\mid_{b=0}= \pi/2$ and $\beta\mid_{b=0}= 0$ has been used here.
For a right triangle on a sphere it holds that:

$$\sin(b) = \tan(a)\cdot\cot(\alpha) \tag{3.54}$$

$$\sin(a) = \tan(b)\cdot\cot(\beta) \tag{3.55}$$

Since a is independent of b, it follows from (3.54) that:

$$\begin{aligned}
&\frac{\partial}{\partial b}\sin(b) = \tan(a)\cdot\frac{\partial}{\partial b}\cot(\alpha)\\
\Rightarrow\quad &\cos(b) = \tan(a)\frac{-1}{sin^2(\alpha)}\frac{\partial\alpha}{\partial b}\\
\Rightarrow\quad &\left.\frac{\partial\alpha}{\partial b}\right|_{b=0} = \frac{-1}{\tan(a)}.
\end{aligned}$$

An analog calculation on (3.55) yields:

$$\left.\frac{\partial\beta}{\partial b}\right|_{b=0} = \frac{1}{\sin(a)}$$

Therefore:

$$\left.\frac{\partial(\delta\Phi_n)}{\partial b}\right|_{b=0} = \lim_{b\to 0}\frac{\Delta(\delta\Phi_n)}{b} = \frac{1}{\tan(a)} - \frac{1}{\sin(a)} = -\tan(a/2) \tag{3.56}$$

Putting it all together:

$$\begin{aligned}
\vec{T}_n^+ &:= \frac{\partial(\delta\Phi_n)}{\partial\vec{p}_{n+1}} = -\tan(a/2)\vec{e}_1\frac{1}{|\vec{s}_n|} = \frac{1}{|\vec{s}_n|}\vec{K}_n\\
\vec{K}_n &:= \frac{\vec{t}_n\times\vec{t}_{n-1}}{1+\vec{t}_{n-1}\cdot\vec{t}_n}
\end{aligned}$$

For the last transformation the addition theorems for trigonometric functions have been used.

For the derivative with respect to $\vec{p}_{n-1}$ the situation is symmetric.

$$\vec{T}_n^- := \frac{\partial(\delta\Phi_n)}{\partial\vec{p}_{n-1}} = \frac{1}{|\vec{s}_{n-1}|}\vec{K}_n$$

For the derivative with respect to $\vec{p}_n$ be aware that $\delta\Phi_n$ only depends on $\vec{p}_{n-1}$, $\vec{p}_n$ and $\vec{p}_{n+1}$ via $\vec{t}_{n-1}$ and $\vec{t}_n$. Therefore:

$$\begin{aligned}
\vec{T}_n := \frac{\partial(\delta\Phi_n)}{\partial\vec{p}_n} &= \frac{\partial(\delta\Phi_n)}{\partial\vec{t}_n}\frac{\partial\vec{t}_n}{\partial\vec{p}_n} + \frac{\partial(\delta\Phi_n)}{\partial\vec{t}_{n-1}}\frac{\partial\vec{t}_{n-1}}{\partial\vec{p}_n} = \frac{\partial(\delta\Phi_n)}{\partial\vec{t}_n}\left(-\frac{\partial\vec{t}_n}{\partial\vec{p}_{n+1}}\right) + \frac{\partial(\delta\Phi_n)}{\partial\vec{t}_{n-1}}\left(-\frac{\partial\vec{t}_{n-1}}{\partial\vec{p}_{n-1}}\right) =\\
&= -\left(\frac{\partial(\delta\Phi_n)}{\partial\vec{p}_{n-1}} + \frac{\partial(\delta\Phi_n)}{\partial\vec{p}_{n+1}}\right)
\end{aligned}$$

Torsion Forces

With these results finding the derivatives of (3.24) is trivial. The assumed degrees of freedom are

$$\boldsymbol{x}_i = \begin{pmatrix} \vec{p}_i \\ \delta\Phi_i \end{pmatrix}.$$

The torsion forces are:

$$\boldsymbol{F}_i^T = -\frac{k^T}{2}\frac{\partial E^T}{\partial \boldsymbol{x}_i} = -k^T \begin{pmatrix} \delta\Phi_{i-1}\vec{T}_{i-1}^{+} + \delta\Phi_i \vec{T}_i + \delta\Phi_{i+1}\vec{T}_{i+1}^{-} \\ -\delta\Phi_{i-1} + \delta\Phi_i - \delta\Phi_{i+1} \end{pmatrix} \tag{3.57}$$

As with all forces, the derivatives with respect to the degrees of freedom are needed for the implicit integration. The simplest derivative is with respect to $\delta\Phi_j$:

$$\frac{\partial \boldsymbol{F}_i^T}{\partial \delta\Phi_j} = k^T \frac{\partial \delta\Phi_i}{\partial \vec{p}_j}$$

The derivatives with respect $\vec{p}_j$ involves deriving the $\vec{T}_i^-$, $\vec{T}_i^+$ and $\vec{T}_i$. These derivatives are found trivially when the derivatives of $\vec{K}_i$ are known.

$$\begin{aligned}
\frac{\partial K_i}{\partial \vec{p}_{i-1}} &= \frac{\partial \vec{s}_{i-1}}{\partial \vec{p}_{i-1}} \frac{\partial}{\partial \vec{s}_{i-1}} \frac{\vec{s}_i \times \vec{s}_{i-1}}{|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}} \\
&= \frac{\partial}{\partial \vec{s}_{i-1}} \frac{\vec{s}_{i-1} \times \vec{s}_i}{|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}} \\
&= \frac{[\vec{s}_i]\left(|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}\right) - \left(\vec{s}_{i-1} \times \vec{s}_i\right)\left(\vec{t}_{i-1}|\vec{s}_i| + \vec{s}_i\right)^T}{\left(|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}\right)^2} \\
&= \frac{[\vec{s}_i] + \vec{K}_i\left(\vec{t}_{i-1}|\vec{s}_i| + \vec{s}_i\right)^T}{|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}}.
\end{aligned}$$

The derivative with respect to $\vec{p}_{i+1}$ works analogously:

$$\begin{aligned}
\frac{\partial K_i}{\partial \vec{p}_{i+1}} &= \frac{\partial \vec{s}_i}{\partial \vec{p}_{i+1}} \frac{\partial}{\partial \vec{s}_i} \frac{\vec{s}_i \times \vec{s}_{i-1}}{|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}} \\
&= \frac{[\vec{s}_{i-1}]\left(|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}\right) - \left(\vec{s}_i \times \vec{s}_{i-1}\right)\left(\vec{t}_i|\vec{s}_{i-1}| + \vec{s}_{i-1}\right)^T}{\left(|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}\right)^2} \\
&= \frac{[\vec{s}_{i-1}] - \vec{K}_i\left(\vec{t}_i|\vec{s}_{i-1}| + \vec{s}_{i-1}\right)^T}{|\vec{s}_i||\vec{s}_{i-1}| + \vec{s}_i\vec{s}_{i-1}}.
\end{aligned}$$

The last missing derivative can be solved with the already known solutions:

$$\frac{\partial \vec{K}_i}{\partial \vec{p}_i} = \frac{\partial \vec{s}_i}{\partial \vec{p}_i}\frac{\partial \vec{K}_i}{\partial \vec{s}_i} + \frac{\partial \vec{s}_{i-1}}{\partial \vec{p}_i}\frac{\partial \vec{K}_i}{\partial \vec{s}_{i-1}} = -\left(\frac{\partial}{\partial \vec{p}_{i+1}} + \frac{\partial}{\partial \vec{p}_{i-1}}\right)\vec{K}_i.$$

3.4. Coupling the thread with the outer world

For the thread to interact with the simulated physical world around it, it has to somehow couple with other objects. The implicit integration allows two types of coupling which can be applied depending on the nature of the interaction.

First the ideas behind these types of couplings are introduced. Later their application to a concrete scenario is explained.

The first type of couplings constraints vertices to a position inhibiting any movement but a prescribed one. This is useful when a vertex is held fixed or due to some other reason constrained to a given position. The second kind of coupling is done by applying forces that vanish when certain constraints are fulfilled. For example, when the thread has moved from the inside to the surface of a solid object.

3.4.1. Constraining vertices to positions

When the thread is grasped by forceps, it is assumed that the external force applied by the instrument is strong enough to hold the vertex at the position dictated by the forceps' position.

Assume the n-th vertex is grabbed. Let D be the set of indices that correspond to the positional degrees of freedom of the n-th vertex (e. g. $\vec{p}_n = (\boldsymbol{x}_{d_1}, \boldsymbol{x}_{d_2}, \boldsymbol{x}_{d_3})$ and $D = \{d_1, d_2, d_3\}$). Then the constraint is enforced by overwriting $B_i(t)$ and $\boldsymbol{a}_i(t)$ from (3.28) with

$$B_i(t) = 0 \qquad \boldsymbol{a}_i(t) = \boldsymbol{x}_{dest,i} - \boldsymbol{x}_i(t)$$

for all $i \in D$. $\boldsymbol{x}_{dest,i}$ denotes the position to which the mass point should be constrained. The new values for these degrees of freedom will be:

$$\boldsymbol{x}_i(t+\Delta t) = \boldsymbol{x}_i(t) + B_i \cdot \boldsymbol{F}_i(t, \boldsymbol{x}(t+\Delta t)) + \boldsymbol{a}_i = \boldsymbol{x}_i(t) + \boldsymbol{a}_i = \boldsymbol{x}_{dest,i}$$

as was required. Of course other degrees of freedom, not describing the position, can also be constrained this way.

3.4.2. External forces

A common approach to prevent object penetration are penalty forces (see e. g. [59, 29]). The idea is to apply a force proportional to the penetration depth. Unfortunately these are difficult to use for interactions with the thread. When suturing the tissue the stress on the thread is big.

The tissue of the blood vessels are simulated using a tetrahedral mesh which is explained by Sismanidis [102]. To prevent the thread from completely penetrating the mesh the proportionality factor for the penalty forces has to be very large. On the other hand, there are situations in which the thread touches the tissue without being pulled and held against the mesh. A high penalty force results in the thread jumping away from the mesh in these situations.

Luckily the implicit integration of the thread provides a tool for better dealing with this situation. It allows for applying forces to the thread which are linear in the position of the thread's vertices. When a vertex penetrates the tissue or some other object, a very large force can be applied that is linear to the penetration depth and vanishes at the obstacle's surface. This way the thread is pushed to the surface but no further and the non-penetration condition can be maintained almost exactly.

As an example, assume a plane defined by

$$P := \{\vec{p} | \, (\vec{p} - \vec{p}_0) \cdot \vec{n} = 0\}.$$

with $\vec{n}, \vec{p}_0 \in \mathbb{R}^3, |\vec{n}| = 1$.

The force

$$\vec{F}_P(\vec{p}_n) := -\alpha \cdot \vec{n}_p \cdot [(\vec{p}_n - \vec{p}_0)\, \vec{n}_p] \tag{3.58}$$

increases with the distance of $\vec{p}_n$ to P and is always directed from $\vec{p}_n$ to the closest point on P. It is also linear in $\vec{p}_n$ and can therefor be applied exactly in the implicit integration.

The idea is still very similar to the concept of penalty forces, but while penalty forces require subtle fine tuning to avoid overshooting as well as unresolved penetration, here the factor α can be set to some high value. A high value ensures that the thread escapes the collision partner even if the force with which it is pressed against it is high. But since the force vanishes at the surface of the object no overshooting can occur.

Yet, there is a disadvantage: An ideal force would push the thread to the outside of the object and vanish for any position where the thread does not penetrate the object (or at least, with a simplification, on the other side of the plane). But the dependence of the force on the position of the thread's vertices is only linear making such a force impossible. In (3.58) the force is chosen in such a way that it attracts the vertices to the surface even if they crossed the plane. This has the undesirable side effect, that the vertices are "glued" to the object's surface. The situation is easily detected by doing an integration step and determine in which direction the resulting force points. One could remove the offending contacts and redo the last integration step. But since all vertices on the thread interact within one time step, the question whether a contact is false can depend on the existence of other contacts. One would have to redo the integration repeatedly, disabling false contacts and (re) enabling correct contacts in between. Unfortunately reintegration also requires recalculation of thread-thread contacts (see section 3.7) which is very costly. Therefore reintegration is not an option.

The undesired effects and how they have been avoided is described in section 3.5.

3.5. Interaction with the Tissue

The tissue of the blood vessels are modeled using a mass spring model on a mesh. This has been done in the work of Sismanidis [102], which is related to this work. As the mesh represents the tissue the terms “tissue” and “mesh” are used synonymously in this section.

3.5.1. Contact handling

For collisions and contacts of the thread with the mesh, the thread is treated as a series of spheres located at the thread’s vertices. Since the lengths of the segments are relatively small compared to the dimensions of the mesh, approximating the thread in this way does not risk missing collisions. Since in a usual situation the collisions happen with a run of vertices, the geometric situation is very similar to the case where the thread is treated as a series of cylinders represented by the segments.

Contact detection

The tunneling of the thread with the mesh needs to be prevented. This will be described in section 3.8 where it will be shown that the the thread can never be inside the mesh. Therefor collision/contact detection has to be done only with the surface triangles. The output is a set of contacts having the following properties:

- The vertex of the thread.
- The surface triangle.
- The penetration depth of the vertex sphere into the triangle.
- The penetration direction, which is the shortest distance along which the vertex has to be moved to resolve the penetration.
- The collision point on the surface of the triangle, which is the last point where triangle and vertex touch each other when the vertex is moved along the penetration direction. This point is represented by it’s barycentric coordinates in

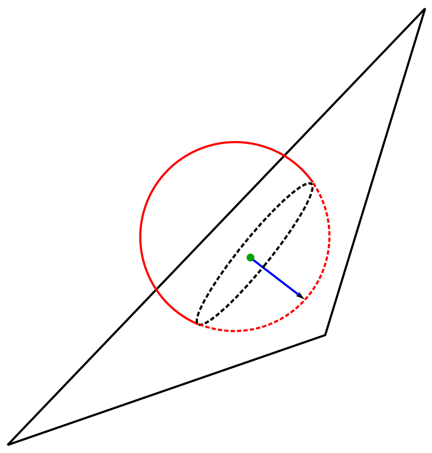


Figure 3.16.: Collision between a surface triangle of the mesh and a vertex of the thread. The blue arrow shows the penetration depth and direction while the green dot is the collision point on the surface of the triangle.

the triangle. For an explanation of barycentric coordinates refer to appendix B.1.

Figure 3.16 illustrates the situation. In the figure, the penetration direction is aligned to the normal of the triangle. This is not necessarily the case when the surface point is on the border of the triangle.

The algorithm for finding these properties works as follows:

1. The collision point and its barycentric coordinates are calculated by finding the closest point on the surface of the triangle to the vertex. Finding the closest point on the surface of triangle to a point is a known problem and described e. g. in appendix B.4.

2. The penetration direction is the normalized vector between the vertex and the collision point.

3. The penetration depth is the length of the vector between the vertex and collision point plus the radius of the thread.

Contact response

Due to the implicit integration, the forces acting on the thread are not known before the thread has been integrated. Therefore the thread has to be integrated before the mesh. Once contacts have been found, response forces are applied to the thread as described in section 3.4.2. The force is set to be proportional to the distance to the plane through the collision point with it's normal equal to the penetration direction. After the thread has been integrated and the forces that acted on it are known, the opposite force is applied to the mesh.

Since forces on the mesh can only be applied to the nodes, the force on the colliding triangle has to be distributed between the nodes of the triangle. An obvious choice for the distribution weights are the barycentric coordinates of the collision points (again, barycentric coordinate are explained in appendix B.1). The barycentric coordinates are an interpolation between the nodes of the triangle. They sum to one and are either positive or zero. At the corners of the triangle the barycentric coordinate for the node at that corner is one while all others are zero.

A more accurate way of applying the force to the mesh would be to subdivide the meshes triangle in such a way that a new node is created at the position where the force is applied. The force could than be applied only to that node. The positions, where forces are applied to the mesh, change in every simulation step. Creating new nodes in every simulation step would result in a unfeasible overhead after a few simulation steps. So the tissue would have to be remeshed often, removing unnecessary nodes and creating nodes where force is applied.

For this work the approximation of applying forces proportional to the barycentric coordinates has been found to suffice to create a visual believable behavior. As the mesh simulation is not part of this thesis, no further investigation in more accurate ways of applying force to the mesh has been done.

Remembering contacts from previous time steps

In a suturing situation the thread can pull the mesh together with a very strong force. If the mesh would not constrain the thread, it could jump to a position that is potentially inside of the mesh resulting in the thread tunneling the mesh.

But when the mesh is integrated the contact forces can move it away from the thread so that the overlap between thread and mesh is resolved and no collision is detected in the next time step. The contact would be removed and when the thread is under strong stress this results in a big leap of the thread resulting in tunneling of the mesh (see figure 3.17).

A common approach for solving this problem is to reset the last integration step, add the contact and reintegrate. As already described in section 3.4.2 , setting a contact affects the complete configuration of the thread after integration. The positions of the vertices can change which means that other contacts may be needed or that old contacts become obsolete by this change. To solve this, one would have to repeat the process of reintegrating and adjusting contacts until no conflicting contacts exist any longer. This requires resolving the thread-thread contacts (3.7) for every reintegration which is computationally expensive. This approach is therefore unfeasible for this application.

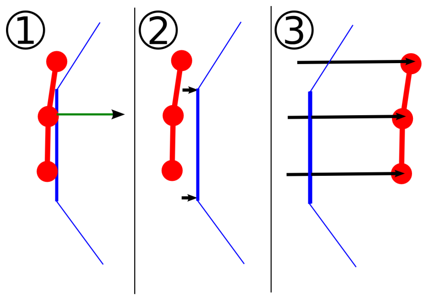


Figure 3.17.: Schematic diagram of the thread tunneling the mesh. The thread is depicted in red while the surface of the mesh is depicted in blue. In the first image the thread is strongly pulled against the mesh producing a force (green arrow) on the mesh. This force causes the mesh to moves away from the thread removing the contact. Without the contact the thread makes a big leap deep into the mesh in the next time step.

The solution suggested in this thesis is to keep contacts between time steps even when thread and mesh do not overlap in the new situation. Only if the contact forces determined by the implicit integration are “gluing” thread and mesh together is the contact removed. This only happens when the thread is pulled away from the mesh.

A negative side effect of this approach is, that the thread is “glued” to the mesh for one time step when pulled away from the mesh. To avoid the mesh following the thread, no force is applied to the mesh in these situations. Because the glue effect exists only for one integration step, it is barely noticeable.

As mesh and thread do not only move in the normal direction of a contact, the saved contacts have to be relocated after every integration step. The vertex of the thread involved in the contact is kept while the corresponding point on the mesh surface is recalculated.

The relocation is accomplished by finding the “locally closest” point to the vertex on the surface of the mesh and setting it to the new collision point. The “locally closest” point is found by walking to the neighboring triangle which is closest to the colliding vertex until no neighboring triangle is closer.

1. Set t_0 to the triangle where the collision point is located and p to the position of the colliding vertex.
2. Set $T := \{t|\text{Triangle } t \text{ shares a corner with } t_0\}$.
3. Set t_1 to the triangle in T whose minimal distance to p is the smallest.
4. If $t_0 = t_1$ then the closest triangle has been found, finish.

5. Continue with step 2 with $t_0 = t_1$.

3.5.2. Sutures

For the purpose of the training simulation the thread in this thesis is made for, a suture is made of two piercings through the blood vessels. The thread goes through these piercings in such a way that when its open ends are knotted it pulls together the blood vessels.

In a first attempt the parts of the thread in the tissue between two penetration points have been modeled by a straight line connection between the piercing points. In the following, we refer to this line as the "suture line". Because of the implicit integration it was possible to apply a force to the vertices forcing them onto the suture line.

As a result of this approach, the thread often left the tissue almost perpendicular to the surface. However, outside the mesh the thread is often parallel to the surface (see figure 3.18(a)). This results in adjacent thread segments being almost orthogonal. When a vertex is pulled outside the suture it is kept on the suture line for the duration of the time step (see figure 3.18(b)). These extreme conditions can result in the vertex moving further outside the mesh than desired leading to unsteady behavior.

To resolve this problem, the thread has been "bent" away from the suture line close to the piercing points in a direction such that the thread moves smoother away form the mesh (see figure 3.19). This way the angle is not as sharp. Yet, maintaining a smooth suture line proved to be difficult especially when the mesh is only a few segment lengths thick.

For these reasons the approach introduced in the next section has been developed.

Portal Forces

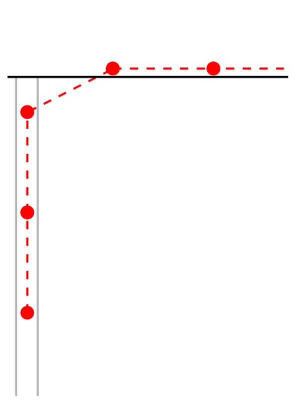
The solution proposed and used in the thesis is not to have any vertices of the thread inside the mesh. Instead the vertex directly before and after the suture are connected through the piercing points.

The idea is reminiscent of the computer game "Portal"[1]. The vertices interact through a "portal spring" that is connected by the piercing points. Figure 3.20 illustrates this. The stretching of this spring is given by the distance of the vertices to the corresponding piercing points.

Let $\vec{s}$ and $\vec{e}$ be the piercing positions. The thread exits the mesh at $\vec{s}$ with $\vec{t_s}$ and at $\vec{e}$ with $\vec{t_e}$.

Let the vertex before the suture be the i-th. Therefor the vertex after the suture is the $(i+1)$-th. To arrive at the forces for the portal spring, an energy is defined

[1]`http://www.valvesoftware.com/games/portal.html`


(a) Thread segments close to a penetration point

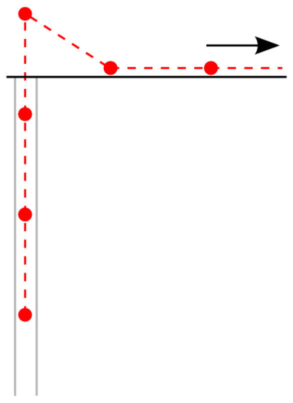

(b) Thread segment pulled out of the mesh

Figure 3.18.: Illustration of problem with modeling a suture by a straight line.

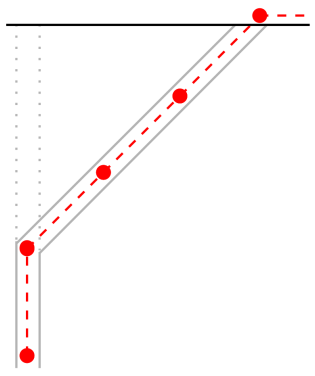


Figure 3.19.: Thread segments in a bend suture line

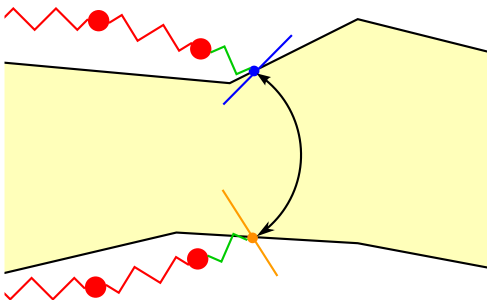


Figure 3.20.: Suture situation with “Portal spring”. The green spring connects the vertices through the piercing points (shown in orange and blue).

similar to the energy of a regular spring. The difference is that stretching of the spring is defined by the sum of the distances from $\vec{s}$ to $\vec{p}_i$ and from $\vec{e}$ to $\vec{p}_{i+1}$.

$$E^P = \frac{k^S}{2}\left(\frac{(\vec{p}_{i+1}-\vec{e})\cdot\vec{t}_e + (\vec{p}_i-\vec{s})\cdot\vec{t}_s}{l_i} - 1\right)^2 \tag{3.59}$$

The forces are found by deriving the energy with respect to the positions:

$$\vec{F}_i^P = -\frac{\partial(E^{Portal})}{\partial\vec{p}_i} = -\frac{k^S}{l_i}\left(\frac{(\vec{p}_{i+1}-\vec{e})\cdot\vec{t}_e + (\vec{p}_i-\vec{s})\cdot\vec{t}_s}{l_i} - 1\right)\cdot\vec{t}_s$$
$$\vec{F}_{i+1}^P = -\frac{\partial(E^{Portal})}{\partial\vec{p}_{i+1}} = -\frac{k^S}{l_i}\left(\frac{(\vec{p}_{i+1}-\vec{e})\cdot\vec{t}_e + (\vec{p}_i-\vec{s})\cdot\vec{t}_s}{l_i} - 1\right)\cdot\vec{t}_e$$

Also needed are the derivatives of these forces:

$$\frac{\partial\vec{F}_i^P}{\partial\vec{p}_i} = -\frac{k^S}{l_i^2}\vec{t}_s\vec{t}_s^T$$
$$\frac{\partial\vec{F}_i^P}{\partial\vec{p}_{i+1}} = -\frac{k^S}{l_i^2}\vec{t}_s\vec{t}_e^T$$
$$\frac{\partial\vec{F}_{i+1}^P}{\partial\vec{p}_i} = -\frac{k^S}{l_i^2}\vec{t}_e\vec{t}_s^T$$
$$\frac{\partial\vec{F}_{i+1}^P}{\partial\vec{p}_{i+1}} = -\frac{k^S}{l_i^2}\vec{t}_e\vec{t}_e^T$$

A definition for $\vec{t}_s$ and $\vec{t}_e$ has not been given yet. The obvious definition is direction from $\vec{s}$ to $\vec{p}_i$ and $\vec{e}$ to $\vec{p}_{i+1}$.

$$\vec{t}_s := \frac{\vec{p}_i - \vec{s}}{|\vec{p}_i - \vec{s}|}$$
$$\vec{t}_e := \frac{\vec{p}_{i+1} - \vec{e}}{|\vec{p}_{i+1} - \vec{e}|}$$

In practice this definition leads to high fluctuations of $\vec{t}_s$ and $\vec{t}_e$ when $\vec{s}$ and $\vec{e}$ move with the mesh. To keep them more steady, they are defined by the direction towards more distance vertices.

$$\vec{t}_s := \frac{\vec{p}_{i-2} - \vec{s}}{|\vec{p}_{i-2} - \vec{s}|}$$
$$\vec{t}_e := \frac{\vec{p}_{i+3} - \vec{e}}{|\vec{p}_{i+3} - \vec{e}|}$$

Sliding through the suture

The portal forces form a "virtual" connection between two vertices. When one vertex moves through its piercing, it should be transported to the other. But how should one define "moving through a piercing"? In practice a piercing point is never directly hit, so what is the difference between moving through a piercing and just moving past it?

The direction vectors $\vec{t}_s$ and $\vec{t}_e$ help to solve this dilemma. They define the direction along which the vertex moves towards or away from the piercing. So "moving through a piercing" means passing the piercing along the direction vector.

Mathematically this is equivalent to defining planes at the piercings with normals $\vec{t}_s$ and $\vec{t}_e$ that include $\vec{s}$ and $\vec{e}$ respectively. A vertex moves through a piercing when it crosses the corresponding plane. This is given when

$$(\vec{p}_i - \vec{s}) \cdot \vec{t}_s < 0.$$

The vertex is moved to a new position $\vec{p}_i'$ that has the same distance to $\vec{e}$ along $\vec{t}_e$ as it penetrates the plane (see figure 3.21).

$$\vec{p}_i' = \vec{e} + \vec{t}_e \cdot d_{in}$$
$$d_{in} := (\vec{s} - \vec{p}_i) \cdot \vec{t}_s$$

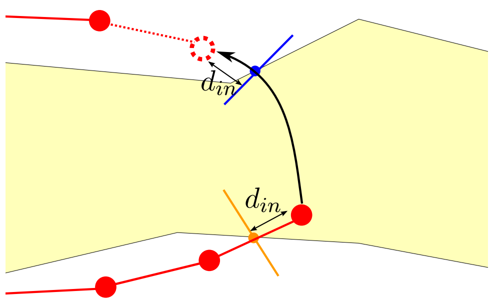


Figure 3.21.: When a vertex slides through a piercing, it is “transported” through the mesh exiting at the corresponding other piercing.

Sideway stability: The portal forces defined by the energy in (3.59) lead to unsteady behavior when the thread is pulled strongly through a piercing point and the pulling direction is not completely aligned with $\vec{t}_s$ or $\vec{t}_e$. The portal forces then allow the attachment of the thread to the piercing plane to slide.

Imagine the thread being pulled on a vertex distant to the piercing plane. The thread will move towards a configuration minimizing the distance between the distant vertex and the piercing plane. Figure 3.22 illustrates the situation. The outcome is an inconsistent situation. $\vec{t}_s$ and $\vec{t}_e$ will be recalculated as a countermeasure in the next time step.

While this process does not cause the thread to diverge by itself, it leads to jumpy behavior which is very undesirable.

As a countermeasure to this effect, the vertex at the piercing points are held on a line through $\vec{s}$ or $\vec{e}$ in direction of $\vec{t}_s$ and $\vec{t}_e$ respectively. The energy for the needed force is given by the quadratic distance between the line and the vertex. In example for the $\vec{s}$ this is

$$E^L := k^S \left[\left(\mathbb{1} - T\left(\vec{t}_s\right)\right) \cdot (\vec{p}_i - \vec{s})\right]^2 .$$

Deriving the forces and their derivatives from this energy is trivial and will not be shown here.

Applying suture forces to the mesh

As in all contact situations, to preserve momentum the total force applied to the mesh and thread in a suture situation must sum to zero. The situation is similar to thread mesh contacts. The forces applied to the thread are not known before the thread has been integrated. Therefore the thread has to be integrated before the forces on the mesh can be found.

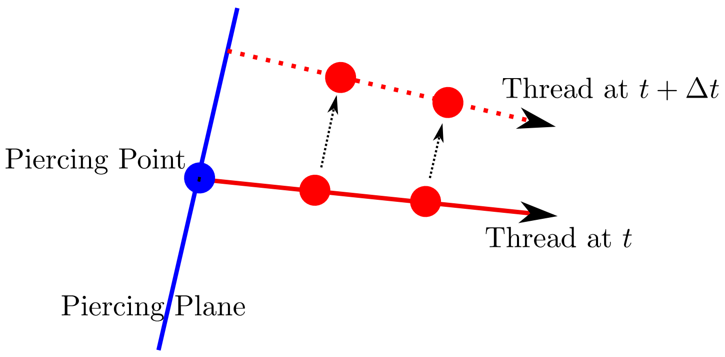


Figure 3.22.: The thread (shown in red) is strongly pulled while attached to a piercing point in a direction non perpendicular to the piercing plane. As a result the touching point slides along the piercing plane making pulling direction perpendicular.

The force on the mesh is applied at two positions: at the two piercings of the suture. The sum of the forces at these two positions is equal to the negative sum of forces applied to the thread due to the suture. There are various ways one could split the total force between these two positions. A simple approach is to apply half of the force to each piercing. As this is very simple and no other approach has proven to work better or realistically, this approach has been used in this thesis.

The piercings are not necessarily at nodes, but at the interior of a triangle. As in section 3.5.1 the force is distributed to the nodes of the triangle with the barycentric coordinates of the piercing.

3.6. Interaction with forceps

This section features interactions with the forceps. There are two types of interactions: First, when the forceps collide with the thread, a collision response similar to the ones with the tissue is performed. Second, when the forceps closes, it grabs the parts of the thread that is between the forceps arms.

3.6.1. Contact handling

The arm of a forceps has been split into 3 geometric objects for contact handling. The main rounded body part is a halved and truncated cone. The tip is represented by a quarter sphere while the cut off side of the cone is extruded with an inclined plane. Figure 3.23 shows the geometric representation of a forceps arm for collision handling. In the following the sum of these geometric objects will be called the “collision object” of a forceps arm.

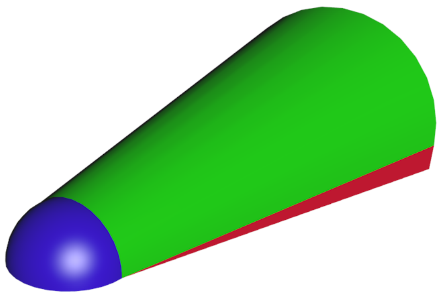


Figure 3.23.: Representing of a forceps arm for collision handling.

As in the collision handling with the tissue, the spheres around the vertices of the thread are used to detect collisions with the forceps. Collisions with a forceps arm and a vertex of the thread are detected by finding the closest point on the surface of the collision object.

To this end, the closest point on all geometric objects of the collision object have to be found. In the following, $\vec{p}$ will be the position of the vertex against which collision is detected. The closest point on the surface will be denoted with $\vec{p}_0$

The collision object is split into 4 regions. Figure 3.24 shows the regions. Depending on the region the vertex is in, only certain points on the surface can be the closest point.

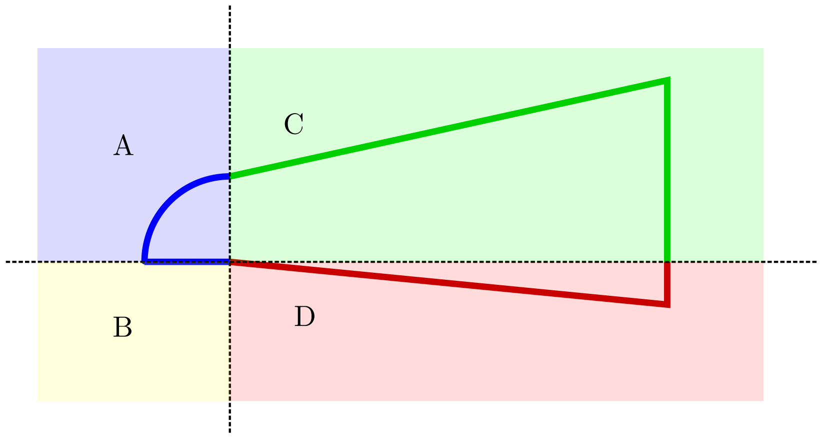


Figure 3.24.: Division of the collision object into regions.

- When the vertex is in region **A** $\vec{p}_0$ can only be on the quarter sphere – assuming the opening angle of the cone is not more than 45°. The closest point on the quarter sphere is found by normalizing the vector from the center of the sphere to $\vec{p}$. The normalized vector is then multiplied by the radius of the sphere and added to its center.

- If $\vec{p}$ is in region **B**, $\vec{p}_0$ is on the half circle at the bottom of the quarter sphere. It is found by projecting $\vec{p}$ onto the plane between region **A** and **B**. If the projected point is situated inside the half circle, it is already the closest point. If not, it is moved toward the center of the circle until it hits the circles border.

- In region **C** the $\vec{p}_0$ must be on the halved and truncated cone. Finding the closest point on the surface of a cone is described in [1]. A short summary is given in appendix B.6.

- When the vertex is in region **D**, $\vec{p}_0$ is on the extended bottom plane of the cone. $\vec{p}_0$ is found by splitting the extension into triangles and finding the closest point on all of these triangles. Appendix B.4 describes how the closest point on a triangle is found.

Once $\vec{p}_0$ has been found, the collision test reduces to testing if $|\vec{p} - \vec{p}_0| < r$ where r is the radius of the thread. If a collision has been detected, response forces are found as described in section 3.4.2. As described in section 3.5.1, contacts are also remembered between integration steps.

3.6.2. Grabbing

For grabbing the technique of enforcing the position of vertices described in section 3.4.1 have been applied.

When the opening of a forceps goes below a certain threshold, all vertices between the forceps arms are grabbed. For this, the positions of the vertices are translated to the local coordinates of the forceps as described in the last section. The vertices between the forceps arms are detected by testing which are situated inside an axis aligned box located between the forceps arms.

For these vertices the local coordinates of the vertices are stored. With the succeeding frames, for every grabbed vertex, the stored local coordinates are translated to global coordinates using the new position of the forceps. The position of the vertex is set to this position and the vertex is set to be fixed to this position as described in section 3.4.1. Since the local forceps coordinates do not change for these vertices, they remain at the same position between the forceps arms.

3.7. Self-interaction of the thread

Interactions between the thread and itself are discussed in this section. To simulate knots convincingly, an accurate collision and contact simulation is necessary. In the literature, one finds the terms “collision” and “contact”, which are closely related. A collision happens over an assumed infinitesimal short moment, leaving the colliding objects in a state where they move apart. A contact is more permanent, it keeps the objects separate but touching. It can be seen as a completely inelastic collision.

Since a knot is a rather static configuration, it is a valid assumption that all collisions are inelastic – as contacts.

In the next section the quantities representing a contact are defined and contact detection is described. Due to the implicit integration when a force is applied to a vertex, it effects all other vertices within the same time step. This is also true for contact forces. As a result, contacts cannot be resolved locally. A global solution is needed to solve all contacts at once.

To find such a global solution, the coupling between contacts must be known. If a force is applied to one contact, how does this effect all other contacts? The coupling will be defined and derived in section 3.7.2.

Once the coupling has been found, a global solution for the contact forces must be found. This problem is similar to the contact problem in rigid body simulations. In section 3.7.2 the requirements for the contact forces and algorithm solving for them are described.

3.7.1. Contact detection

To detect collisions and contacts of a thread with itself or other threads, a rigid representation has to be found for which intersections between threads can be calculated. The canonical representation for a segment is a cylinder whose center line corresponds to the segments center line. Its radius is the radius of the thread.

When two successive cylinders are not perfectly aligned, there is a gap between the cylinders bases. These gaps are filled by spheres with the same radius around the vertices. See figure 3.25.

Pairs of segments have to be tested for contact. A simple approach would be to test every segment against every other segment. The complexity of this brute force approach is $O(n^2)$ where n is the number of segments. The computational burden for this approach is to high for the number of segments needed in the simulation. Various approaches for reducing this complexity exist. Most of them partition the space, allowing to do collision detection only for objects that are close enough to each other for a collision to be possible. In this thesis, the spatial hashing from Teschner et al. [111] has been applied. It is explained a little more detailed in section 4.1.4.

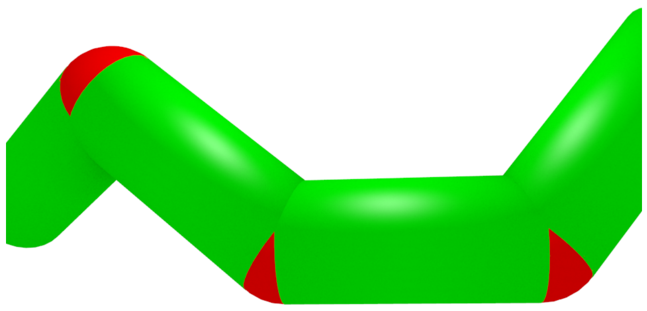


Figure 3.25.: Rigid representation of the thread. Segments are shown in green while vertices are red.

Contact description

A contact consists of

- two contact positions $\vec{c}_1, \vec{c}_2$ which represent the position on the center line of the thread parts in contact. They are the point on the corresponding center line that is closest to the other center line.
- a separation vector $\vec{s} := \vec{c}_2 - \vec{c}_1$ connection the contact positions.

The separation vector is later used to apply the contact force to the segments. Because a contact force is always orthogonal to the surface, $\vec{s}$ must be orthogonal to the surface at both contact positions. In the following, this requirement will be called the "orthogonality constraint of $\vec{s}$". The contact positions $\vec{c}_1, \vec{c}_2$ are described by the indexes $a_{\{1,2\}} \in \{1 \ldots N\}$ of the vertex or segment involved in the contact and a parameter $\mu_{\{1,2\}} \in [0, 1]$ describing the position along the segment. The relation is:

$$\vec{c}_n = (1 - \mu_n) \cdot \vec{p}_{a_n} + \mu_n \cdot \vec{p}_{a_n+1} \qquad n \in \{1, 2\}$$

In summary a contact c is described by a 4-tuple

$$C := (a_1, a_2, \mu_1, \mu_2) \tag{3.60}$$

Contact types

A contact must fulfill the following criterias:

- The contact positions must be less than the sum of the radii apart. This means $|\vec{s}| \leq r_1 + r_2$ where r_1 and r_2 are the radii of the threads in contact.
- $\vec{s}$ must be orthogonal to the surface of the thread where the line between the contact positions intersects. This must be true for both contact partners.

The first criteria is easily tested once the contact positions have been found. In practice it is eased a little by testing $|\vec{s}| \leq 1.1\,(r_1 + r_2)$. This way, positions where the threads are almost in contact are also tested. The contact resolution does not apply any force to contacts which remain resolved by themselves. The second criteria must be tested differently depending on whether the shortest line between the contact positions intersects with a cylinder or a sphere.

For a **segment-segment** contact first, the closest points between the center lines of the segments are found (appendix B.5 explains how). If both of them are on the interior of the segments, the orthogonality constraint is fulfilled (see figure 3.26). If the positions are close enough a contact is found.

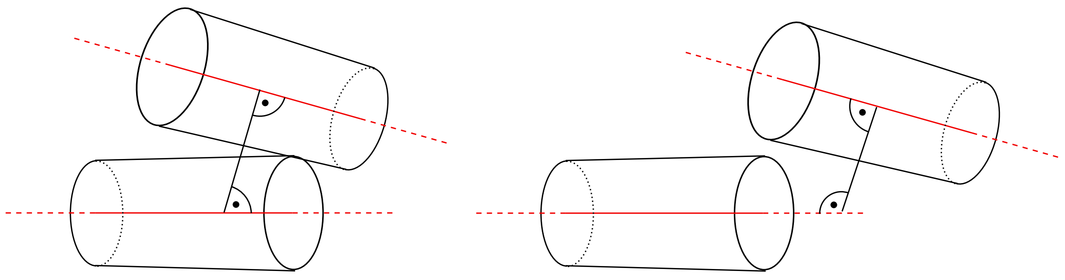


Figure 3.26.: Contacts between segments.
Left: The closest points are within a segment, the segments collide.
Right: For the lower segment, the closest point is outside. No collision between the segments takes place.

If the closest points are not in the interiors of the center lines, one or both contact positions might still be set to the ends of the center lines (to a vertex position). This type of contact will be called a **segment-vertex** contact. The closest point of one segment to a vertex of the other segment is found (appendix B.3 explains how). For the orthogonality constraint to be fulfilled at the segments side, the contact position must be again on the interior of the segment. On the vertex side the contact position is set to the vertex position. As a result, the line between the contact positions always intersects the spheres surface where $\vec{s}$ is orthogonal. But the sphere must also be the threads surface at that position. This is the case when $\vec{s}$ points away from the bottom planes of both adjacent cylinders (see figure 3.27).

Mathematically this means:

$$\begin{aligned} 0 &\geq \vec{s} \cdot \vec{t}_{i+1} \\ 0 &\geq \vec{s} \cdot (-\vec{t}_i) \end{aligned} \tag{3.61}$$

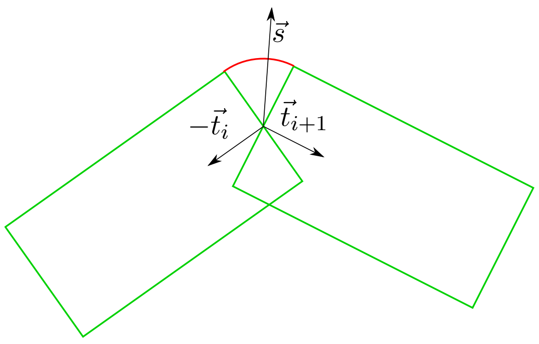


Figure 3.27.: For a contact involving a vertex, $\vec{s}$ must point away from the bottom plane of both adjacent cylinders.

A **vertex-vertex** contact is found if condition 3.61 is fulfilled for both setting both contact partners when setting both contact positions to vertices..

While these are the obvious cases, there is another case for **segment-vertex** and **vertex-vertex** contacts which only becomes important when the thread radius is close to the segment length. The segment length has to be short enough to ensure a flexible behavior of the thread. Figure 3.28 pictures the situation. One of two segments is bended around the other, but no collision is detected.

This occurs because the surface of the thread is not smooth in the inner side of a bend between adjacent segments.

The criteria in (3.61) can also be formulated as follows: For the vertex to become the contact partner, the closest point to the other contact partner on the center line of the adjacent segments must be outside the interior towards the vertex. It must be lossened to account for the problem pictured in figure 3.28: The vertex can also become the contact partner when the closest point to the other contact partner on the center line of the adjacent segments is outside the interior away from the vertex.

3.7.2. Contact handling

Once contacts have been detected, the corresponding contact forces have to be found. All contacts are assumed to be inelastic. The main task of the contact forces is to maintain non-penetration constraints at all contact points.

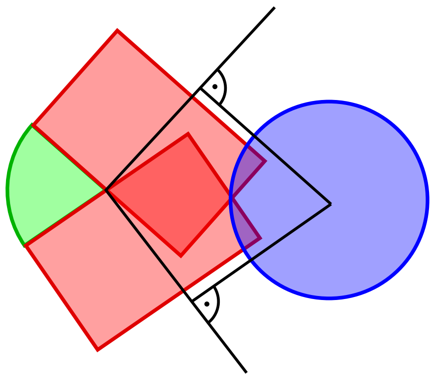


Figure 3.28.: A thread segment upright on the pictures plane (blue) collides two segments that are bended around it (red). Also collisions occur, no collision is detected between the blue and the red segments because the contact points are outside the red segments center lines.

It would be possible to exploit the implicit integration scheme and define implicit forces at a contact point that vanishes when the non-penetration constraint is fulfilled. There are two obvious approaches to this:

- The first approach is defining a contact force that vanishes when the contact positions have a distance of twice the thread radius in the separation direction, similar to section 3.5 and 3.6. The resulting force would be linear in all involved vertex positions and can therefore be expressed exactly as an implicit force. Unfortunately this creates forces on the vertices in contact that depend on the position of vertices at completely different positions on the thread. This introduces non zero entries outside the bands of the banded matrix for the implicit integration (see section 3.2.2). Doing so would raise the complexity for solving the system from $O(n)$ to $O(n^3)$, which is not acceptable.

- To address this problem, forces for both contact partners that are independent from each other could be defined. The forces would simply push the threads away from the contact position (which is fixed in space for a time step). While the complexity problem would vanish, the vertices involved in the contact would now be fixed to the position where the contact occurs. Even when both threads move as a whole, vertices involved in a contact would stick to a fixed position in space, leading to unrealistic physical behavior.

At this point it is interesting to see that while the second approach is very similar to the approach used for contacts with the mesh, it still works there. The difference is that in a thread-mesh collision, one of the collision partners (the mesh) is not simulated with the implicit integration. It is therefore not constrained by the minimum of the potential of the implicit forces and can move freely. With mesh-thread contacts, the mesh "guides" the movement while the thread influences it by applying forces to the mesh. It is also important that the mesh has a much bigger mass then the thread and moves slower. On the time scale that the thread moves, the mesh can be seen as almost fixed.

In addition to the discussed problems above, both approaches do not describe the contact forces correctly. They do not vanish when the contacts are separated while friction forces are difficult to model. If the threads happen to be separated by other effects (i.E. other contacts), contacts that should have vanished "stick" the thread together.

Coupling of contacts

After excluding unfavorable methods a more in depth view for the requirements of the contact forces is desirable. Since there may exist multiple contacts at once, the contacts will be given an index and denoted by

$$C_i := (a_{1,i}, a_{2,i}, \mu_{1,i}, \mu_{2,i}) \qquad\qquad 1 \leq i \leq M$$

Where M is the number of contacts. On every contact a contact force $\vec{F}_i^c$ is applied. Given the movement of the adjacent mass points, the relative contact movement is

$$\begin{aligned}\Delta\vec{c}_i &:= \Delta\vec{c}_2 - \Delta\vec{c}_1 \\ &= (1-\mu_{2,i}) \cdot \Delta\vec{p}\,'_{a_{2,i}} + \mu_{2,i} \cdot \Delta\vec{p}\,'_{a_{2,i}+1} - (1-\mu_{1,i}) \cdot \Delta\vec{p}_{a_{1,i}} - \mu_{1,i}\Delta\vec{p}_{a_{1,i}+1}.\end{aligned} \tag{3.62}$$

Having a set of contacts $\{C_i\}$, a set of contact forces $\{\vec{F}_i^c\}$ must be found. Because contacts are coupled, the contact forces have to be found globally solving all contacts at the same time. The exact meaning of "solving a contact" will be elaborated in the subsection "contact resolution" below.

Firstly the relation between an external force applied to a vertex and the effect on the position of all vertices will be analyzed. Assume an additional force $\vec{E}_n$ is applied to the n-th vertex. The force is "additional" because it is added to any other force already applied to that vertex. How does this force change the movement of the i-th vertex? $\Delta\vec{p}_i$ is a function of $\vec{E}_n$. Based on this $\delta_{n,i}$ is defined by:

$$\vec{\delta}_{n,i}(\vec{E}_n) := \Delta\vec{p}_i(\vec{E}_n) - \Delta\vec{p}_i(0). \tag{3.63}$$

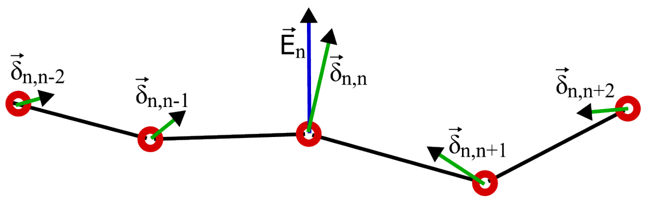


Figure 3.29.: An external force (blue arrow) results in displacements of all mass points (green arrows).

Figure (3.29) illustrates the relation between the defined quantities.

Note that $\vec{\delta}_{n,i}(\vec{E}_n)$ can be defined as above, and is in particular independent of $\Delta\vec{p}_i(0)$. This is concluded from the fact, that the $\Delta\vec{p}_i$ are linear in all forces applied to any vertices. This also means that $\vec{\delta}_{n,i}$ is linear in $\vec{E}_n$. It will be illustrated later by deriving the linear relations.

$\vec{\delta}_{n,i}$ could be found simply by setting the $\vec{E}_n$ and doing the integrating step. $\vec{\delta}_{n,i}$ is then found by applying (3.63). Unfortunately this requires a lot of integrations which have a complexity of $O(N)$. This is not doable due to limited computational power.

The relation between $\vec{\delta}_{n,i}$ and $\vec{E}_n$ needs to be found by plugging them into (3.31). To do this they must be extended to all degrees of freedom – not only the position.

Let $\boldsymbol{\delta}_{n,i}$ be $\vec{\delta}_{n,i}$ extended to all degrees of freedom of the i-th vertex while $\boldsymbol{\delta}_n \in \mathbb{R}^N$ (without the second index) is the same for all degrees of freedom in the thread. $\boldsymbol{E}_n$ is defined from $\vec{E}_n$ by setting the forces on the remaining degrees of freedom for the n-th vertex to zero. $\hat{\boldsymbol{E}}_n \in \mathbb{R}^N$ additionally has a vanishing force for all other vertices.

The quantities can now be plugged into (3.36):

$$M(\Delta\boldsymbol{x}(0) + \boldsymbol{\delta}_n) = \boldsymbol{a} + B(\boldsymbol{F} + \hat{\boldsymbol{E}}_n). \tag{3.64}$$

Subtracting (3.36) from (3.64) yields:

$$M\boldsymbol{\delta}_n = B\hat{\boldsymbol{E}}_n. \tag{3.65}$$

Or, written more explicitly:

$$D_i\boldsymbol{\delta}_{n,i} + O_i^{+}\boldsymbol{\delta}_{n,i+1} + O_i^{++}\boldsymbol{\delta}_{n,i+2} + O_i^{-}\boldsymbol{\delta}_{n,i-1} + O_i^{--}\boldsymbol{\delta}_{n,i-2} = \begin{cases} B_n\boldsymbol{E}_n \text{ if } i = n \\ 0 \text{ otherwise} \end{cases} \tag{3.66}$$

Since (3.65) is linear, the relation between $\vec{\delta}_{n,i}$ and $\vec{E}_n$ is also linear.

It will now be shown, that for $i < n$ there are $M_i^+, M_i^{++} \in \mathbb{R}^F$ such that

$$\boldsymbol{\delta}_{n,i} = M_i^+ \boldsymbol{\delta}_{n,i+1} + M_i^{++} \boldsymbol{\delta}_{n,i+2}. \tag{3.67}$$

Remember the intent of the $+$, $++$, $-$ and $--$ from equation (3.31). The same notation is followed here. The $+$ and $++$ in M_i^+ and M_i^{++} indicate the relation to previous vertex and the vertex before the previous respectively.

Equation (3.67) will be shown using mathematical induction deriving M_i^+ and M_i^{++} along the way. The base clause with $i = 1$ is easy to show. Starting from (3.66) one gets:

$$\begin{aligned} & D_1 \boldsymbol{\delta}_{n,1} + O_1^+ \boldsymbol{\delta}_{n,2} + O_1^{++} \boldsymbol{\delta}_{n,3} = 0 \\ \Rightarrow\quad & \boldsymbol{\delta}_{n,1} = -D_1 \left(O_1^+ \boldsymbol{\delta}_{n,2} + O_1^{++} \boldsymbol{\delta}_{n,3} \right) = M_1^+ \boldsymbol{\delta}_{n,2} + M_1^{++} \boldsymbol{\delta}_{n,3} \end{aligned}$$

with

$$M_1^+ = -D_1^{-1} O_n^+ \qquad\qquad M_1^{++} = -D_1^{-1} O_n^{++} \tag{3.68}$$

The induction step to $i = 2$ is treated differently from the following steps. The relation for $\boldsymbol{\delta}_{n,2}$ can be found utilizing the just found result for $\boldsymbol{\delta}_{n,1}$:

$$\begin{aligned} & D_2 \boldsymbol{\delta}_{n,2} + O_2^+ \boldsymbol{\delta}_{n,3} + O_2^{++} \boldsymbol{\delta}_{n,4} = 0 \\ \Rightarrow\quad & D_2 \boldsymbol{\delta}_{n,2} + O_2^- \left(M_1^+ \boldsymbol{\delta}_{n,2} + M_1^{++} \boldsymbol{\delta}_{n,3} \right) + O_2^+ \boldsymbol{\delta}_{n,3} + O_2^{++} \boldsymbol{\delta}_{n,4} = 0 \\ \Rightarrow\quad & \left(D_2 + O_2^- M_1^+ \right) \boldsymbol{\delta}_{n,2} + \left(O_2^- M_1^{++} + O_2^+ \right) \boldsymbol{\delta}_{n,3} + O_2^{++} \boldsymbol{\delta}_{n,4} = 0 \\ \Rightarrow\quad & \boldsymbol{\delta}_{n,2} = M_2^+ \boldsymbol{\delta}_{n,3} + M_2^{++} \boldsymbol{\delta}_{n,4} \end{aligned}$$

with

$$\begin{aligned} M_2^+ &= -\left(D_2 + O_2^- M_1^+ \right)^{-1} \cdot \left(O_2^+ + O_2^- M_1^{++} \right) \\ M_2^{++} &= -\left(D_2 + O_2^- M_1^+ \right)^{-1} \cdot \left(O_2^{++} \right) \end{aligned} \tag{3.69}$$

M_2^+ and M_2^{++} are defined in terms of M_1^+ and M_1^{++}. M_i^+ and M_i^{++} will also be defined in terms of M_{i-1}^+ and M_{i-1}^{++}. For the induction step with $i > 2$ let M_j^+, M_j^{++}

be known for $j < i$. Then starting from (3.66):

$$\begin{aligned}
& D_i\boldsymbol{\delta}_{n,i} + O_i^{+}\boldsymbol{\delta}_{n,i+1} + O_i^{++}\boldsymbol{\delta}_{n,i+2} + O_i^{-}\boldsymbol{\delta}_{n,i-1} + O_i^{--}\boldsymbol{\delta}_{n,i-2} = 0 \\
\Rightarrow\quad & D_i\boldsymbol{\delta}_{n,i} + O_i^{--}\left(M_{i-2}^{+}\boldsymbol{\delta}_{n,i-1} + M_{i-2}^{++}\boldsymbol{\delta}_{n,i}\right) \\
& \qquad + O_i^{-}\left(M_{i-1}^{+}\boldsymbol{\delta}_{n,i} + M_{i-1}^{++}\boldsymbol{\delta}_{n,i+1}\right) \\
& \qquad + O_i^{+}\boldsymbol{\delta}_{n,i+1} + O_i^{++}\boldsymbol{\delta}_{n,i+2} \qquad = 0 \\
\Rightarrow\quad & (D_i + O_i^{--}M_{i-2}^{++} + O_i^{-}M_{i-1}^{+})\boldsymbol{\delta}_{n,i} \\
& \qquad + O_i^{--}M_{i-2}^{+}\left(M_{i-1}^{+}\boldsymbol{\delta}_{n,i} + M_{i-1}^{++}\boldsymbol{\delta}_{n,i+1}\right) \\
& \qquad + \left(O_i^{-}M_{i-1}^{++} + O_i^{+}\right)\boldsymbol{\delta}_{n,i+1} + O_i^{++}\boldsymbol{\delta}_{n,i+2} \qquad = 0 \\
\Rightarrow\quad & (D_i + O_i^{--}M_{i-2}^{++} + O_i^{-}M_{i-1}^{+} + O_i^{--}M_{i-2}^{+}M_{i-1}^{+})\boldsymbol{\delta}_{n,i} \\
& \qquad + \left(O_i^{--}M_{i-2}^{+}M_{i-1}^{++} + O_i^{-}M_{i-1}^{++} + O_i^{+}\right)\boldsymbol{\delta}_{n,i+1} \\
& \qquad + O_i^{++}\boldsymbol{\delta}_{n,i+2} \qquad = 0 \\
\Rightarrow\quad & \boldsymbol{\delta}_{n,i} \quad = M_i^{+}\boldsymbol{\delta}_{n,i+1} + M_i^{++}\boldsymbol{\delta}_{n,i+2}
\end{aligned}$$

where

$$\begin{aligned}
M_i^{+} &= K_i^{+}\left(O_i^{+} + T_i^{+}M_{i-1}^{++}\right) & M_i^{++} &= K_i^{+}O_i^{++} \\
K_i^{+} &:= -\left(D_i + O_i^{--}M_{i-2}^{++} + T_i^{+}M_{i-1}^{+}\right)^{-1} & T_i^{+} &:= O_i^{-} + O_i^{--}M_{i-2}^{+}
\end{aligned} \tag{3.70}$$

For $i > n$ one can do similar calculations, but then the recursion starts at the last vertex (the N-th), and gets:

$$\boldsymbol{\delta}_{n,i} = M_i^{-}\boldsymbol{\delta}_{n,i-1} + M_i^{--}\boldsymbol{\delta}_{n,i-2} \tag{3.71}$$

with

$$\begin{aligned}
M_N^{-} &= -D_N^{-1}O_N^{-} \\
M_N^{--} &= -D_N^{-1}O_N^{--} \\
M_{N-1}^{-} &= -\left(D_{N-1} + O_{N-1}^{+}M_N^{-}\right)^{-1} \cdot \left(O_{N-1}^{-} + O_{N-}^{+}M_N^{--}\right) \\
M_{N-1}^{--} &= -\left(D_{N-1} + O_{N-1}^{+}M_N^{-}\right)^{-1} \cdot \left(O_{N-1}^{--}\right) \\
M_i^{-} &= K_i^{-}\left(O_i^{-} + TM_{i+1}^{--}\right) \\
M_i^{--} &= K_i^{-}O_i^{--} \\
T_i^{-} &:= O_i^{+} + O_i^{++}M_{i+2}^{-} \\
K_i^{-} &:= -\left(D_i + O_i^{++}M_{i+2}^{--} + TM_{i+1}^{-}\right)^{-1}
\end{aligned}$$

The last calculations missing are $\boldsymbol{\delta}_{n,n}$, $\boldsymbol{\delta}_{n,n-1}$ and $\boldsymbol{\delta}_{n,n+1}$. Actually $\boldsymbol{\delta}_{n,n}$ and one of its neighbors would be enough, but since the situation is symmetric there is no harm in calculating both. At least one of the neighbors is needed, because the calculations of $\boldsymbol{\delta}_{n,i}$ always needs two previous values to be known already. Note the requirement $i > n$ and $i < n$ cannot be dropped because for $i = n$ the right hand side of (3.66) does not vanish for this case.

First $\boldsymbol{\delta}_{n,n+1}$ will be found. In (3.71) for $i = n+1$, $\boldsymbol{\delta}_{n,n-1}$ is replaced using 3.67, which reveals the formula for $\boldsymbol{\delta}_{n,n+1}$:

$$\begin{aligned} \boldsymbol{\delta}_{n,n+1} &= M^{-}_{n+1}\boldsymbol{\delta}_{n,n} + M^{--}_{n+1}\left(M^{+}_{n-1}\boldsymbol{\delta}_{n,n} + M^{++}_{n-1}\boldsymbol{\delta}_{n,n+1}\right) \\ \Rightarrow \quad \boldsymbol{\delta}_{n,n+1} &= N^{+}_{n}\boldsymbol{\delta}_{n,n} \\ N^{+}_{i} &:= \left(1 - M^{--}_{i+1}M^{++}_{i-1}\right)^{-1}\left(M^{-}_{i+1} + M^{--}_{i+1}M^{+}_{i-1}\right). \end{aligned} \tag{3.72}$$

An analogous calculation can be done to find $\boldsymbol{\delta}_{n,n-1}$:

$$\begin{aligned} \boldsymbol{\delta}_{n,n-1} &= N^{-}_{n}\boldsymbol{\delta}_{n,n} \\ N^{-}_{i} &:= \left(1 - M^{--}_{i+1}M^{++}_{i-1}\right)^{-1}\left(M^{+}_{i-1} + M^{++}_{i-1}M^{-}_{i+1}\right). \end{aligned} \tag{3.73}$$

This leaves $\boldsymbol{\delta}_{n,n}$ to be calculated. The starting point is again (3.66), but this time the right hand side is not zero:

$$D_n\boldsymbol{\delta}_{n,n} + O^{+}_{n}\boldsymbol{\delta}_{n,n+1} + O^{++}_{n}\boldsymbol{\delta}_{n,n+2} + O^{-}_{n}\boldsymbol{\delta}_{n,n-1} + O^{--}_{n}\boldsymbol{\delta}_{n,n-2} = \boldsymbol{E}_n$$

$\boldsymbol{\delta}_{n,n+2}$ and $\boldsymbol{\delta}_{n,n-2}$ are replaced using (3.67) and (3.71), and the whole set of equations is reordered, grouping the factors of $\boldsymbol{\delta}_{n,n}$, $\boldsymbol{\delta}_{n,n-1}$ and $\boldsymbol{\delta}_{n,n+1}$.

$$\begin{aligned} &\left[D_n + O^{++}_{n}M^{--}_{n+2} + O^{--}_{n}M^{++}_{n-2}\right]\boldsymbol{\delta}_{n,n} \\ &+ \left[O^{-}_{n} + O^{--}_{n}M^{+}_{n-2}\right]\boldsymbol{\delta}_{n,n-1} \\ &+ \left[O^{+}_{n} + O^{++}_{n}M^{-}_{n+2}\right]\boldsymbol{\delta}_{n,n+1} \\ =&A_n\boldsymbol{\delta}_{n,n} + B_{n,1}\boldsymbol{\delta}_{n,n-1} + B_{n,2}\boldsymbol{\delta}_{n,n+1} = \boldsymbol{E}_n \end{aligned} \tag{3.74}$$

where

$$\begin{aligned} A_i &= D_i + O^{++}_{i}M^{--}_{i+2} + O^{--}_{i}M^{++}_{i-2} \\ B_{i,1} &= O^{-}_{i} + O^{--}_{i}M^{+}_{i-2} \\ B_{i,2} &= O^{+}_{i} + O^{++}_{i}M^{-}_{i+2} \end{aligned}$$

Combining (3.74), (3.72) and (3.73) yields the equation for $\boldsymbol{\delta}_{n,n}$ as follows.

$$\begin{aligned} & A_n \boldsymbol{\delta}_{n,n} + B_{n,1} N_n^- \boldsymbol{\delta}_{n,n} + B_{n,2} N_n^+ \boldsymbol{\delta}_{n,n} = \boldsymbol{E}_n \\ \Rightarrow \quad & \boldsymbol{\delta}_{n,n} = \left(A_n + B_{n,1} N_n^- + B_{n,2} N_n^+ \right)^{-1} \boldsymbol{E}_n \end{aligned} \tag{3.75}$$

The coupling Matrices between vertices:

While (3.67), (3.71) , (3.72) (3.73) and (3.75) define relations between subsets of $\boldsymbol{\delta}_{n,i}$, it is now necessary to find matrices $G_{n\to i}$ defined by

$$\boldsymbol{\delta}_{n,i} = G_{n\to i} \cdot \boldsymbol{E}_n$$

(3.72), (3.73) and (3.75) already directly define $G_{n\to n}$ and $G_{n\to n\pm 1}$:

$$G_{n\to n} = \left(A_n + B_{n,1} N_n^- + B_{n,2} N_n^+ \right)^{-1} \qquad G_{n\to n+1} = N_n^+ \qquad G_{n\to n-1} = N_n^-$$

The rest follows a recursive definition. If e. g. $G_{n\to i-1}$ and $G_{n\to i-2}$ are given, then:

$$G_{n\to i} = M_i^- G_{n\to i-1} + M_i^{--} G_{n\to i-2}. \tag{3.76}$$

Analogously, if $G_{n\to i+1}$ and $G_{n\to i+2}$ are given, then:

$$G_{n\to i} = M_i^+ G_{n\to i+1} + M_i^{++} G_{n\to i+2}. \tag{3.77}$$

To resolve a set of contacts simultaneously, pairwise couplings between a given set of vertices are needed. Using (3.76) one needs to iterate between all possible pairs of vertices. To reduce the computational burden, it would be desirable if less iterations between the outermost vertices would be needed.

And indeed, a reduction to only 2 iterations is possible. To reach it, a relation between different $G_{n\to i}$ is needed. A direct relation between a single arbitrary $G_{n\to i}$ cannot be found, but if successive coupling matrices are paired, a relation between these pairs can be found.

To put it in mathematical terms, there is a matrix $H_{j\to i}$ with $i \geq j$, such that:

$$\begin{pmatrix} G_{n\to i} \\ G_{n\to i+1} \end{pmatrix} = H_{j\to i} \begin{pmatrix} G_{n\to j} \\ G_{n\to j+1} \end{pmatrix} \tag{3.78}$$

$H_{j\to i}$ is split into 4 parts:

$$H_{j\to i} = \begin{pmatrix} H^1_{j\to i} & H^2_{j\to i} \\ H^3_{j\to i} & H^4_{j\to i} \end{pmatrix}$$

Starting with $j = i$, $H_{j\to i}$ is defined inductively. When looking at (3.78), the initial case with $i = j$ is trivial:

$$\begin{pmatrix} H^1_{j\to i} & H^2_{j\to i} \\ H^3_{j\to i} & H^4_{j\to i} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

The induction step $i \to i+1$ is:

$$\begin{pmatrix} H^1_{j\to i+1} & H^2_{j\to i+1} \\ H^3_{j\to i+1} & H^4_{j\to i+1} \end{pmatrix} = \begin{pmatrix} H^3_{j\to i} & H^4_{j\to i} \\ M^-_{i+1} H^3_{j\to i} + M^{--}_{i+1} H^1_{j\to i} & M^-_{i+1} H^4_{j\to i} + M^{--}_{i+1} H^2_{j\to i} \end{pmatrix}$$

Finding the matrix $H_{j\to i}$ for $i < j$ is done analogously.

With these new tools and observations, all $G_{n\to i}$ can be calculated with only two iterations over the thread's vertices in a given index set I, one in upward direction and one in downward direction. The upward direction calculates the $G_{n\to i}$ for $i \geq n$ while the downward direction calculates those with $i \leq n$. The resulting algorithm is given in listing 1. It has to be run for $d = 1$ and $d = -1$.

Algorithm 1 Finding $G_{n\to i}$ for $n, i \in I$. $d \in \{1, -1\}$ indicates the direction in which I is iterated

```
sort $I$
$j \leftarrow$ Nil                                  ▷ $j$ is the index of the last vertex that was visited
for $i \in I$ in ascending order if $d = 1$ and descending order if $d = -1$ do
    $G_{i\to i} \leftarrow \left(A_i + B_{i,1}N_i^- + B_{i,2}N_i^+\right)^{-1}$
    if $d = 1$ then
        $G_{i\to i+1} \leftarrow N_i^+$
    else
        $G_{i\to i-1} \leftarrow N_i^-$
    end if
    if $j =$ Nil then
        continue          ▷ Do not go into section below, it is not for the first element
    end if
    calculate $H_{j\to i}$
    for all $n \in I, d \cdot n < i$ do
        $\begin{pmatrix} G_{n\to i} \\ G_{n\to i+d} \end{pmatrix} \leftarrow H_{j\to i} \begin{pmatrix} G_{n\to j} \\ G_{n\to j+d} \end{pmatrix}$
    end for
    $j \leftarrow i$
end for
```

While the algorithm is still quadratic in the number of constraints, the complexity is reduced from $O(M^2N)$ to $O(M^2 + N)$ when compared to the direct approach.

Coupling matrix between contacts

Assume two contacts $C_i := (a_{1,i}, a_{2,i}, \mu_{1,i}, \mu_{2,i})$ and $C_j := (a_{1,j}, a_{2,j}, \mu_{1,j}, \mu_{2,j})$. First the effect of a given Contact force $\vec{F}_i^c$ will be examined. The force is split up and applied to the nodes adjacent to the segment in contact proportional to their distance to the contact position. For the first thread the applied forces are:

$$\begin{aligned} \Delta\vec{E}_{a_{1,i}} &= \vec{F}_i^c \cdot (1 - \mu_{1,i}) \\ \Delta\vec{E}_{a_{1,i}+1} &= \vec{F}_i^c \cdot \mu_{1,i} \end{aligned} \tag{3.79}$$

While for the second thread they are:

$$\begin{aligned} \Delta\vec{E}'_{a_{2,i}} &= -\vec{F}_i^c \cdot (1 - \mu_{2,i}) \\ \Delta\vec{E}'_{a_{2,i}+1} &= -\vec{F}_i^c \cdot \mu_{2,i} \end{aligned} \tag{3.80}$$

Since the force is constant over the time step, $\frac{\partial \vec{F}_i^c}{\partial \vec{x}_i} = 0$. Momentum is preserved because these forces sum to zero.

Because of the implicit integration, applying a force to one vertex affects the future position of all vertices in the thread. This means that contacts are coupled even when they do not involve the same vertices.

How does $\vec{F}_i^c$ affect C_j? In particular, how does it effect the relative contact movement $\Delta\vec{c}_j$? The coupling $G_{C_i \to C_j}$ is defined by the relation between $\vec{F}_i^c$ and the relative contact movement $\Delta\vec{c}_j$. Without any contact forces there is already a relative movement of contacts originating from other external and internal forces. This movement is found by integrating the thread without any contact forces. These movements will be denoted by $\Delta\vec{c}_i^0$. With this quantity, the definition of $G_{C_i \to C_j}$ is:

$$\Delta\vec{c}_j - \Delta\vec{c}_j^0 = G_{C_i \to C_j} \vec{F}_i^c \tag{3.81}$$

To derive $G_{C_i \to C_j}$, remember the definition of $\Delta\vec{c}_j$ from (3.62). With the index set paired with factors

$$P((i_1, i_2, \mu_1, \mu_2)) := \{(i_1, 1-\mu_1), (i_1+1, \mu_1), (i_2, \mu_2 - 1), (i_2+1, -\mu_2)\}$$

(3.62) can be rewritten as:

$$\Delta\vec{c}_j = \sum_{(i,\mu) \in P(C_j)} \mu \Delta\vec{p}_i$$

Assuming the external forces $\vec{E}_n$ are applied (the index n corresponds to the vertices to which the force is applied) are added by contacts, this can be written as:

$$\Delta\vec{c}_j - \Delta\vec{c}_j^0 = \sum_{(i,\mu) \in P(C_j)} \mu \left(\sum_n \vec{\delta}_{n,i}(\vec{E}_n) \right)$$

The contact force $\vec{F}_i^c$ will be applied to the vertices in the first contact by:

$$\Delta\vec{E}_j := \nu \vec{F}_i^c \qquad\qquad \text{for } (j, \nu) \in P(C_i).$$

Putting these together reveals:

$$G_{C_i \to C_j} := \sum_{\substack{(i,\mu) \in P(C_j) \\ (j,\nu) \in P(C_i)}} \mu\nu G_{i \to j}$$

Moving from 2 to $M \in \mathbb{N}$ contacts $\{C_i\}$, $i = 1 \ldots M$, and the corresponding contact forces $\vec{F}_i^c$. To arrive at the equations solving the contacts, a few quantities have to be defined.

The contact force vector is defined by

$$\boldsymbol{F}_{\{C_i\}} := \left(\left(\vec{F}_1^c\right)^T, \left(\vec{F}_2^c\right)^T, \ldots\right)^T. \tag{3.82}$$

A vector for the relative movements before contact forces have been applied is defined by:

$$\Delta\boldsymbol{c}^0_{\{C_i\}} := \left(\left(\Delta\vec{c}_1^{\,0}\right)^T, \left(\Delta\vec{c}_2^{\,0}\right)^T, \ldots\right)^T,$$

and also a vector for the relative movements after the contact forces have been applied:

$$\Delta\boldsymbol{c}_{\{C_i\}} := \left(\left(\Delta\vec{c}_1\right)^T, \left(\Delta\vec{c}_2\right)^T, \ldots\right)^T.$$

With matrix Q given by

$$Q := \begin{pmatrix} G_{c_1\to c_1} & G_{c_2\to c_1} & \cdots \\ G_{c_1\to c_2} & G_{c_2\to c_2} & \cdots \\ \cdots & \cdots & \cdots \end{pmatrix}$$

the relation between these quantities is given by

$$\Delta\boldsymbol{c}_{\{C_i\}} = Q \cdot \boldsymbol{F}_{\{C_i\}} + \Delta\boldsymbol{c}^0_{\{C_i\}} \tag{3.83}$$

This relation has to be solved respecting the condition given in the next section.

Normal contact force constraint

The contact force as well as the relative contact movement are split into a normal and a tangential part. The normal is aligned to the separation direction $\vec{s}_i$ while the tangential part is the projection onto the plane with normal $\vec{s}_i$. The normal part is responsible for keeping the separation constraint while the tangential parts model friction.

Let $\vec{n}_i := \vec{s}_i / |\vec{s}_i|$ the normalized separation vector. There is an orthonormal coordinate system $\{\vec{n}_i, \vec{f}_i, \vec{g}_i\}$ with the normal and 2 friction directions. $\vec{F}_i^c, \Delta\vec{c}_i$ and $\Delta\vec{c}_i^{\,0}$ are projected into this coordinate system:

$$\begin{aligned} \alpha_i &:= \vec{F}_i^c \cdot \vec{n}_i & \beta_i &:= \vec{F}_i^c \cdot \vec{f}_i & \gamma_i &:= \vec{F}_i^c \cdot \vec{g}_i \\ a_i &:= \Delta\vec{c}_i \cdot \vec{n}_i & b_i &:= \Delta\vec{c}_i \cdot \vec{f}_i & c_i &:= \Delta\vec{c}_i \cdot \vec{g}_i \\ a_i^0 &:= \Delta\vec{c}_i^{\,0} \cdot \vec{n}_i & b_i^0 &:= \Delta\vec{c}_i^{\,0} \cdot \vec{f}_i & c_i^0 &:= \Delta\vec{c}_i^{\,0} \cdot \vec{g}_i \end{aligned}$$

There are several conditions defining the resolution of a contact. Firsty, the penetration must be resolved:

$$a_i - r_1 - r_2 \geq 0 \tag{3.84}$$

where r_1 and r_2 are the radii of the colliding threads.

The contacts are resolved by applying a normal force which has to push the contacts apart.

$$\alpha_i \geq 0 \tag{3.85}$$

The normal force must either resolve the penetration exactly ($a_i - r_1 - r_2 = 0$) or it must be zero $\alpha_i = 0$. This can be expressed by a complimentary condition:

$$\alpha_i(a_i - r_1 - r_2) = 0 \tag{3.86}$$

Equation (3.84) - (3.86) form a linear complimentary problem (short LCP, see e. g. [32]) which can be solved with e. g. Lemkes algorithm. The equations can also be reformulated as a quadratic programming (QP) problem. When no friction is involved, all forces have only a component in normal direction. Reducing (3.83) by projection all quantities on the normal parts, it becomes:

$$\boldsymbol{a} = A \cdot \boldsymbol{\alpha} + \boldsymbol{a^0}$$

with

$$\boldsymbol{a} := \begin{pmatrix} a_1 \\ a_2 \\ \vdots \end{pmatrix} \quad \boldsymbol{\alpha} := \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \end{pmatrix} \quad \boldsymbol{a^0} := \begin{pmatrix} a_1^0 \\ a_2^0 \\ \vdots \end{pmatrix} \quad A := \begin{pmatrix} \vec{n}_1^T G_{C_1 \to C_1} \vec{n}_1 & \vec{n}_1^T G_{C_2 \to C_1} \vec{n}_2 & \dots \\ \vec{n}_2^T G_{C_1 \to C_2} \vec{n}_1 & \vec{n}_2^T G_{C_2 \to C_2} \vec{n}_2 & \dots \\ \dots & \dots & \dots \end{pmatrix}$$

Since (3.84) and (3.85) imply that 0 is the minimum of (3.86), the problem can be written as:

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha} \left(A \cdot \boldsymbol{\alpha} + \boldsymbol{a^0} \right) \qquad \text{subject to } A \cdot \boldsymbol{\alpha} + \boldsymbol{a^0} \geq \boldsymbol{r_1} + \boldsymbol{r_2} \text{ and } \boldsymbol{\alpha} \geq \boldsymbol{0}$$

which is a quadratic programming (QP) problem. [9] argue that moving to a QP problem discards the information in (3.86) and therefore makes the problem more general and thus harder. The problem will also not be an LCP anymore when adding general friction constraints and cannot be transformed into a QP directly. These are the reasons why the QP view will not be followed any further.

Friction force constraint

The formulation so far does not involve friction constraints, which are also important for a physical plausible behavior – especially when simulating knots. The Coulomb laws for friction are as follows:

1. The magnitude of the friction force is smaller (static friction) or equal (kinetic friction) to the normal force times the coefficient of friction.
2. Static friction occurs only when the friction force is strong enough to prevent any tangential movement.
3. The force of friction is always directed in the opposing direction of movement.

For simplicity in many simulations it is assumed, that the coefficient of friction for kinetic and static friction are equal. This assumption will also be made here. The first friction constraint translated to a formula is:

$$\beta_i^2 + \gamma_i^2 \leq \mu^2 \alpha_i^2 \tag{3.87}$$

where $\mu \geq 0$ is the coefficient of friction. In case of kinetic friction, the limiting case $\beta_i^2 + \gamma_i^2 = \mu^2 \alpha_i^2$ is hit. In case of static friction the inequality is not an equality.
When plotting this constraint as a function of α_i a cone results, which is called the friction cone (see figure 3.30).

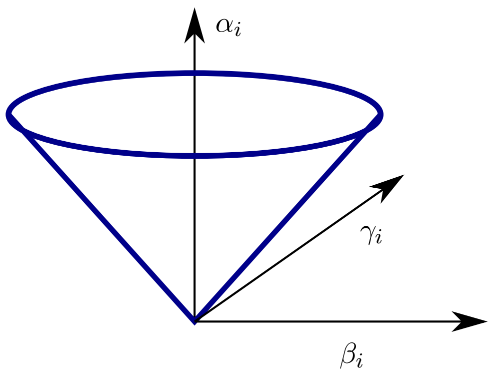


Figure 3.30.: The friction cone resulting from (3.87)

When adding this constraint, the problem is not a linear complementary problem, but a nonlinear complementary problem (NCP). Many LCP algorithm cannot be applied anymore. There are several ways in the literature to modify this constraint keeping it compatible with the LCP formulation. Usually they approximate the constraint in 3.87. The Box constraint constraints both directions independently:

$$-\mu\alpha_i \leq \beta_i \leq \mu\alpha_i$$
$$-\mu\alpha_i \leq \gamma_i \leq \mu\alpha_i$$

A constraint like this has the unfortunate side effect that the tangential friction forces are not invariant under rotations around $\vec{n}_i$. This raises the question of how to choose $\vec{f}_i$ and $\vec{g}_i$ and how to ensure that this choice does not oscillate between integration steps. An oscillation would lead to unsteady behavior of the simulation, which is very undesirable. Other works (e. g. [106, 4]) reduce this invariance by approximating the friction cone with a polygon (see figure 3.31). The LCP formulation in this form has a much higher memory demand (more constraint equations) than the NCP formulation with (3.87). Fortunately, the algorithms applied in this thesis are able to solve the NCP problem.

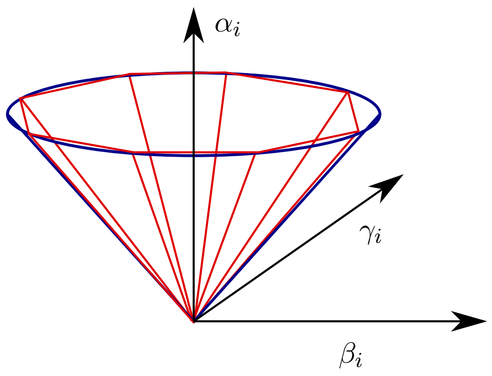


Figure 3.31.: Polygonal approximation of the friction cone.

The second friction law states, that the friction force must be on the border of the friction cone or the tangential movement must be zero. This can be stated as a complimentary condition:

$$(\beta_i^2 + \gamma_i^2 - \mu^2\alpha_i^2) \cdot (b_i^2 + c_i^2) = 0 \tag{3.88}$$

which is again not a LCP condition. Again, it can be made LCP compliant by stating it for both directions separately.

The last friction law can be stated as:

$$\begin{pmatrix}\beta_i \\ \gamma_i\end{pmatrix} = -\lambda \begin{pmatrix} b_i \\ c_i \end{pmatrix} \qquad \lambda \geq 0 \tag{3.89}$$

Baraff [9] simplifies this constraint by stating that the friction force be at least partially opposed to the tangential acceleration:

$$\begin{pmatrix}\beta_i\\ \gamma_i\end{pmatrix} \cdot \begin{pmatrix}b_i\\ c_i\end{pmatrix} \leq 0 \tag{3.90}$$

which, in difference to (3.89) is LCP compatible. Note that (3.90) follows from (3.89).

Contact Resolution

Several algorithms exists to resolve LCPs. Chapter 2 lists related works and algorithms for solving the contact problem.

Iterative approaches have the advantage that they can be interrupted any time and output approximations of the final result. This bears the advantage that the computational burden is predictable, e. g. by fixing the number of iterations. In real time application this is important to maintain the real time constraint.

The projected Gauss-Seidel (PGS) (see e. g. [20]) is an iterative algorithm for solving LCPs. The idea of the projected Gauss-Seidel is simple. In essence, every constraint is solved in turn using the current result from the other constraints. Assuming a function "solveConstraint" that satisfies the conditions from section 3.7.2 for a single contact, the PGS applied to the contact problem is stated in algorithm 2.

Algorithm 2 The projected Gauss-Seidel, applied to the contact problem.

```
for all $\{1 \dots NUM\_ITERATIONS\}$ do
    for all $i \in \{1 \dots |\{C_i\}|\}$ do
        $\Delta \boldsymbol{c} \leftarrow \sum_j Q_{i,j} \cdot \boldsymbol{F}_{\{C_j\}} + \Delta \boldsymbol{c}_0(C_i)$
        $\boldsymbol{F}_{C_i} \leftarrow \text{solveContact}(C_i, \Delta \boldsymbol{c})$
    end for
end for
```

In this thesis, the PGS approach has been taken by providing a function that solves (3.84), (3.85), (3.86), (3.87), (3.88) and (3.89). (3.89) is solved only approximately by assuming that the direction of the kinetic friction force is the same as a static friction force that is not constrained by (3.87).

Another advantage of the PGS is that it can be "warm-started" as suggested in [20]. The idea is that the algorithm is initialized with an approximation of the correct solution (the initial guess). Assuming the configuration of the thread and contacts does not change drastically between integration step, the solution of the last invocation provides a usable initial guess.

Silcowitz-Hansen et al. [101] improves the convergence rate of the PGS. The problem is viewed as a nonlinear complementary problem (NCP) and a Fletcher-Reeves type nonlinear nonsmooth conjugate gradient (NNCG) type method is applied. Both PGS and NNCG have been tested for the problem and NNCG has been found to output better results in most cases. Silcowitz-Hansen et al. [101] do no line search in the gradient direction, because their experiments show that nothing is gained by a line search. For this thesis also no line search is done, but the step size is limited so that it does not step out of the area where (3.84) is fulfilled. This avoids some cases where in the original NNCG algorithm the error increases drastically between iterations.

The stopping criteria for the iterative solver has been chosen to a fixed number of iterations. This has the advantage of making the time consumption of the solver predictable, reducing variation in time consumption between update steps.

It should be noted that the problem could also have been solved by adding the contact constraints to the integration matrix of the implicit integration scheme and solving it as an LCP problem. This way the contact couplings would not have to be calculated. But that would level the integration problem from solving a banded linear system to solving a sparse LCP problem. In difference to the LCP problem needed for solving the contacts in a separated step as it has been described in this chapter, the integration is a problem of much bigger size. Therefore, the advantage of calculating the coupling between contacts is that the LCP part can be extracted from the integration and can be solved in a separated step where the problem has a much smaller size.

Secondary contacts

A common problem with contact detection and resolution for rigid bodies is that secondary contacts or collisions occur. Due to existing contacts being resolved, new contacts are created at other positions which only occur because the existing contacts change the movement of the objects.

A common solution to this problem consists in testing for contacts and resolving them repeatedly until all occurring contacts are resolved. Unfortunately this is not possible in this work because it would require rerunning the LCP solver which is not feasible for a realtime application. The problem is reduced by using the "low mass integrator" which will be described in section 4.1.2). Due to this integrator the thread has no impulse and does not bounce of itself which would increase the risk of a secondary contact.

Also the problem of secondary contacts mainly occurs when the thread is in a "dynamic" situation, meaning it moves with a certain speed. In this situations physical inaccuracy does not harm the believability for a human observer as much as it does in static situation.

Still, missed contacts are a big problem when they cause the thread to tunnel through itself and thereby change the topology. This can cause knots to “magically” disappear which is an unacceptable behavior. To resolve this, tunneling is prevented in a additional step described in section 3.8.4.

Keeping a list of active contacts

There is an observation which allows to boost the performance of the constraint solver. The contact detection also adds contacts when the segments do not yet overlap but are still in a small distance from each other. This is done to prevent tunneling effects. If the contact is added falsely, it does not change the result because constraint solver will set the contact force to zero.

This and other effects cause many situations where there are many inactive contacts during constrain solving. As the contact force of these inactive contacts is zero, they do not affect the other contacts. When calculating $\sum\limits_{j} Q_{i,j} \cdot \boldsymbol{F}_{\{C_j\}} + \Delta \boldsymbol{c}_0(C_i)$ in algorithm 2 these contacts can be skipped. To efficiently only iterate over the contacts that are active, a list of active contacts is kept.

3.8. Continuous collision detection

The implicit integration allows integration of stiff forces with a large time step. But due to the computational demand of the implicit integration, a smaller time step cannot be chosen. This large time steps come with a burden. The thread is very thin – a typical diameter of a microsurgery thread is $0,02mm$ – and the movement of a vertex within one time step can be a multiple of the threads diameter. When the collision detection does not detect moments where the thread intersects with itself, the tissue or the forceps might miss situations where the thread “tunnels”. In this case a collision is missed.

The contradictions that come with such topological changes go beyond physical plausibility. A knot can dissolve, which is of course fatal when trying to make the suture knot. The thread can detach itself from the forceps or dissolve a suture by tunneling through the tissue. For these reasons, tunneling must be prevented under all circumstances.

How can tunneling be detected? Assume a thread with vertices positions $\vec{p}_i(t)$. The integration sets the positions at times $n\Delta t$ where $n \in \mathbb{N}_0$. Between these positions a linear interpolation is assumed. So for $n\Delta t \leq t \leq (n+1)\Delta t$ the vertex positions are:

$$\vec{p}_i(t) = \vec{p}_i(n\Delta t) + \frac{t - n\Delta t}{\Delta t} \left((\vec{p}_i((n+1)\Delta t) - \vec{p}_i(n\Delta t) \right) .$$

For the thread to tunnel, there must be a time t_0 where

- a vertex is on the surface of a triangle.
- a vertex is on the surface of a forceps.
- the center lines of two segments touch.

In this section the counter measurements to tunneling are discussed. The tunneling is prevented by either moving the involved objects back to the position where the tunneling happened or moving the vertices of the thread out of the object along the normal of the surface.

In the following, the tunneling prevention strategies for tissue forceps and thread are discussed separately. But first it will be discussed how to detect when 4 points are planar (on a common plane) in 3D, as this is required for thread-tissue as well as thread-thread tunneling.

3.8.1. Detecting when 4 moving points are planar

Assume 4 points which move linearly with time:

$$\vec{q}_1(t) = \vec{q}_1(0) + t\vec{v}_1 \quad \vec{q}_2(t) = \vec{q}_2(0) + t\vec{v}_2 \quad \vec{q}_3(t) = \vec{q}_3(0) + t\vec{v}_3 \quad \vec{q}_4(t) = \vec{q}_4(0) + t\vec{v}_4$$

The goal of this section is to detect if the 4 points are on the same plane within a time-interval $[t_1, t_2]$. When they are, it should also be determined at what time-point (there are potentially 3) the planarity occurs and what the earliest moment is.

$\vec{q}_1(t_0), \vec{q}_2(t_0)$ and $\vec{q}_3(t_0)$ form a triangle with normal:

$$\vec{N} := \frac{(\vec{q}_1(t_0) - \vec{q}_2(t_0)) \times (\vec{q}_1(t_0) - \vec{q}_3(t_0))}{|(\vec{q}_1(t_0) - \vec{q}_2(t_0)) \times (\vec{q}_1(t_0) - \vec{q}_3(t_0))|}$$

For $\vec{q}_4(t_0)$ to lie in the same plane, its distance to any point in the triangle projected onto $\vec{N}$ must vanish. See figure 3.32.

In other words, one has to solve:

$$[(\vec{q}_1(t_0) - \vec{q}_2(t_0)) \times (\vec{q}_1(t_0) - \vec{q}_2(t_0))] \cdot (\vec{q}_4(t_0) - \vec{q}_1(t_0)) = 0$$

which is a cubic system with potentially 3 solutions. The smallest solution in the time interval $[t_1, t_2]$ is the earliest moment of planarity.

For an algorithm to solve a cubic system refer to e.g. Press [88, chapter 5.6].

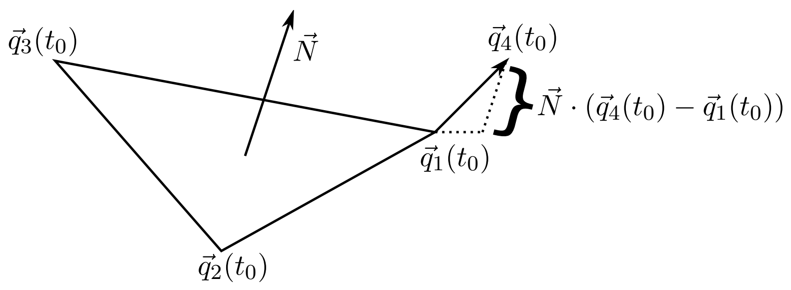


Figure 3.32.: The 4 points are planar, if $\vec{N} \cdot (\vec{q}_4(t_0) - \vec{q}_1(t_0)) = 0$

3.8.2. Preventing tunneling of the thread with the tissue

To test if the thread is tunneling into the mesh representing the tissue, a vertex of the thread has to go through a surface triangle of the mesh. A necessary condition for this is that there is a time point in the time interval of the integration step, in which the vertex is in the plane of the triangle. The planarity test from the last section can be used to determine this.

If the vertex lies in the same plane in the inside of the triangle, a tunneling is detected. To test this, the barycentric coordinates (see appendix B.1) of the planar point are calculated and tested to determine if they are all in the interval $[0, 1]$. If this is the case, the point is in the inside of the triangle.

For resolving the tunneling, it is assumed that the tissue is vastly heavier than the thread. Therefore the vertex of the thread is moved to resolve the inconsistency created by the tunneling. It is moved along the triangles normal to the outside of the mesh.

3.8.3. Preventing tunneling of the thread with the forceps

The collision object of the forceps is represented by a cone, a sphere and planes (see section 3.6). Doing continuous collision detection the same way as with the tissue would require the difficult task of detecting intersections of a moving point (the vertex) and a moving cone, sphere and plane.

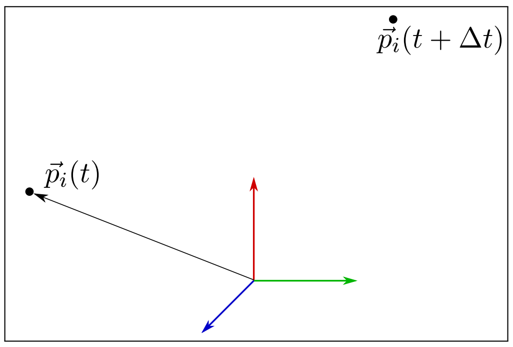
To avoid this, it is assumed that the forceps and thread move alternating. This involves detecting collisions of the moving cone with the vertex which is still difficult. For that reason the vertex position is held fixed in the coordinate system of the forceps when the forceps moves.

Every forceps arm has a coordinate system that moves with it. Since the arms of the forceps have a static shape, the coordinate system can be defined in a way that the corresponding forceps arm does not move or change in it.

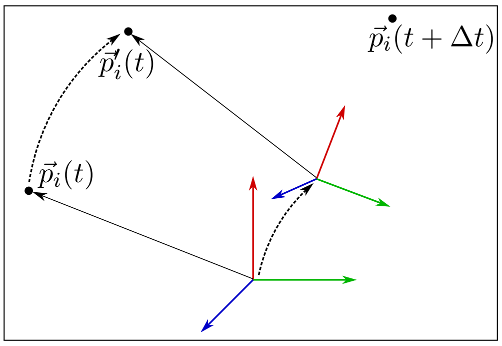
Assume a vertex with position $\vec{p}_i$ is not inside the collision object of a forceps arm.

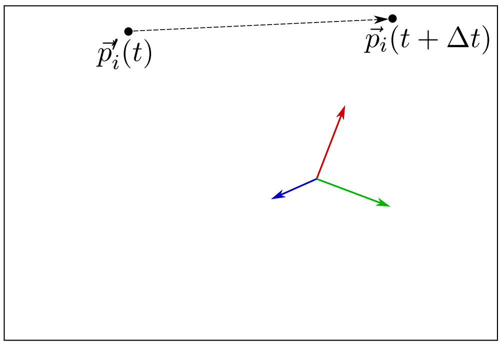
Now the arm is moved. Let $\vec{p}_i'$ be the position which has the same representation in the new coordinate system of the forceps arm as $\vec{p}_i$ in the old coordinate system. Then $\vec{p}_i'$ is not inside the collision object after the arm has been moved.

The vertex is not really moved with the forceps arm. It is just assumed that the old position of the vertex is $\vec{p}_i'$ when doing continuous collision detection with a forceps arm.



(a) $\vec{p}_i(t)$ represented in the forceps coordinate system





(b) Moving the forceps coordinate system and thereby moving $\vec{p}_i(t)$ to $\vec{p}_i'(t)$

(c) Movement of vertex due to integration step

Figure 3.33.: Movement of the i-th vertex for continuous collision detection with the forceps

Figure 3.33 illustrates the process. In figure 3.33(a) $\vec{p}_i(t)$ is represented in the coordinate system of a forceps arm before it moves. Then the arm moves and the vertex position moves to $\vec{p}_i'(t)$ with the coordinate system. Then continuous collision detection is done with the movement of the vertex from $\vec{p}_i'(t)$ to $\vec{p}_i(t+\Delta t)$.

Now that the process has been described, it will be put into more mathematical terms. There is an affine transformation $M_t : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ from the global coordinate system to the coordinate system of the forceps at time point t. $\vec{p}_i'(t)$ is defined by:

$$\vec{p}_i'(t) = \left(M_{t+\Delta t}^{-1} \circ M_t\right)(\vec{p}_i(t))$$

The movement from $\vec{p}_i'(t)$ to $\vec{p}_i(t+\Delta t)$ is assumed to be linear. To detect if and when the vertex "hits" the collision object of the forceps arm, it is required to find intersections between a line segment and a sphere, a cone and a triangle. In appendix B.7 and B.8 the algorithm for finding intersections between a line segment and a sphere and cone are described.

When a collision is detected, the vertex is moved back to the time point where the collision occurs.

3.8.4. Preventing tunneling of the thread with itself

Let i and j be the indices of the vertices for which continuous collision detection has been done. A necessary condition for the center line to touch at t_0 is, that $\vec{p}_i(t_0), \vec{p}_{i+1}(t_0), \vec{p}_j(t_0)$ and $\vec{p}_{j+1}(t_0)$ are in the same plane. The test for the time-points fulfilling these requirements is described above in section 3.8.1. The sufficient condition for the thread to tunnel is that at the moment where the vertices are planar, the center lines of the segments touch.

When a tunneling has been detected and the vertex has been moved back to the moment of contact, the segments are pushed slightly apart so that the non-continuous collision detection can find the correct separation direction. This can cause other tunnelings. So the two segments have to be retested.

4. Results

4.1. Implementing an anastomosis training module in MicroSim

The thread has been embedded into MicroSim (see also [56]), a prototype for a training simulator for microsurgical tasks. A training module has been created in which the anastomosis of two blood vessels is performed.

4.1.1. MicroSim setup

The setup consists of a wooden box which has been covered with black velvet from the inside. Four cameras are located in the top corners of the interior. The forceps, with which the virtual operation is performed, are equipped with 3 colored markers each. The cameras can detect the markers and thereby triangulate the position and opening of the forceps. This system has been developed by Schuppe [96]. Figure 4.1 and 4.2 show the inside of the box as well as the forceps with markers.

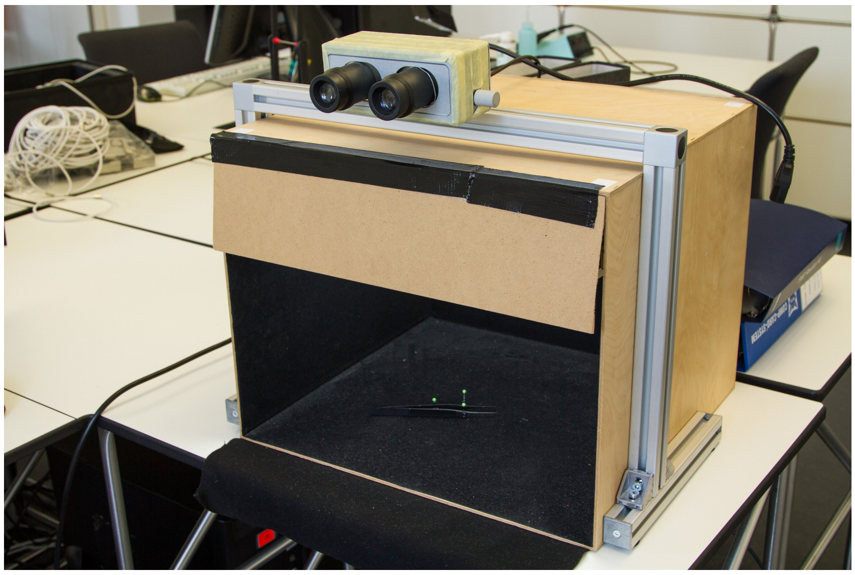


Figure 4.1.: Microsim setup

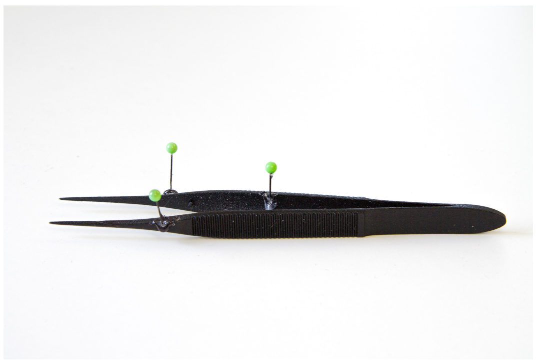


Figure 4.2.: Forceps with markers

The tissue simulation is based on a mass spring simulation which has been developed by Sismanidis [102].

The output of the simulation is displayed on a stereo display through which the trainee can observe the scene similar to when looking through a stereo microscope used in real surgery. Figure 4.3 shows how the aperture is used.

4.1.2. Low mass Integration scheme

The implicit integration from the last section must be based on an integration scheme. When extending the Newton equation of motion with friction, that is linear to the velocity of the motion, it becomes:

$$m\frac{\partial^2 \boldsymbol{x}}{\partial t^2}(t) + \mu\frac{\partial \boldsymbol{x}}{\partial t}(t) = \boldsymbol{F}(\boldsymbol{x}(t)) \tag{4.1}$$

Where μ controls the strength of the friction. Assuming the mass m is very low and the object moves in a viscous fluid – which makes μ high – it can be assumed, that

$$m\frac{\partial^2 \boldsymbol{x}}{\partial t^2}(t) \ll \mu\frac{\partial \boldsymbol{x}}{\partial t}(t) \tag{4.2}$$

allowing (4.1) to be simplified to

$$\mu\frac{\partial \boldsymbol{x}}{\partial t}(t) \approx \boldsymbol{F}(\boldsymbol{x}(t)). \tag{4.3}$$

An object following this equation of motion has no impulse. There is no state in equation (4.3). The movement only depends on the forces in the current configuration and not on its history. The configuration always moves towards equilibrium. The very light mass of the thread justifies this approximation. Visual inspection of

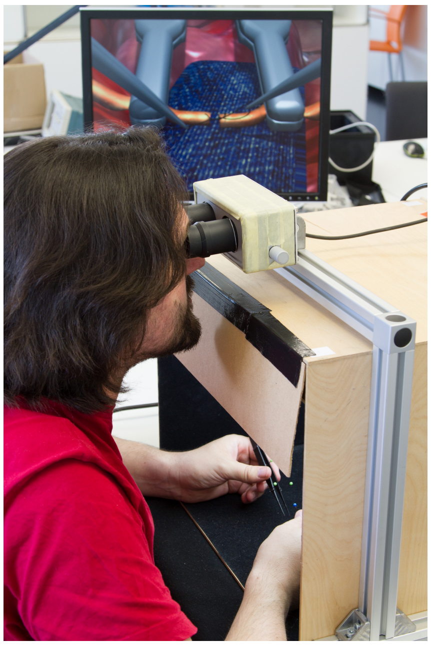


Figure 4.3.: A trainee using MicroSim

the behavior of light threads also supports the approximation. When brought out of equilibrium they transform quickly to a new equilibrium state.

The approximation is desirable because it allows the thread to move more steadily. This is important for the simulation of knots. With unsteady behavior, knots can dissolve spontaneously in an unrealistic way. When pushing the threads apart in a contact situation an impulsive behavior would result. This causes the segments in contact to move further apart in the next time step and to terminate the contact.

Discretizing the approximated equation of motions gives

$$\Delta \boldsymbol{x} = \nu \boldsymbol{F}(\boldsymbol{x}(t))$$

where $\nu := \frac{1}{\mu}$ is the mobility.

For the implicit integration in this thesis, this can be realized by setting

$$\boldsymbol{a} = \boldsymbol{0} \qquad B = \nu \mathbb{1}$$

This integration scheme has been used in all simulations in the results.

4.1.3. Removing the torsional degree of freedom from the implicit integration

Bergou et al. [13] argue that torsional waves move much faster through a thread than bending waves. Based on this, they update the torsion decoupled from the remaining degrees of freedom. For a thread with no intrinsic bending, this means that the twist angle is evenly distributed between vertices for which the twisting is fixed. In other words $\delta\Phi_i$ is equal for all vertices between the fixed vertices.

For a pair of fixed vertices that have no fixed vertex in between, the total twist angle is calculated. Then it is divided by the number of vertices in between and $\delta\Phi_i$ is set to this value for all of them. When this is done, $\delta\Phi_i$ can be removed as a degree of freedom calculated by the implicit integration. Forces on the other degrees of freedom due to twisting are still calculated the same way as when $\delta\Phi_i$ is not removed as a degree of freedom handled by the implicit integration.

In section 3.7 self interaction of the thread is treated. For this, a number of matrices with dimensions equal to the number of degrees of freedom for a vertex have to be inverted. When $\delta\Phi_i$ is a degree of freedom, there are a total of 4 degrees of freedom at every vertex. If not, there are only 3. Inverting a 3×3 matrix takes much less steps than inverting a 4×4 matrix. Removing the torsion degree of freedom from the implicit integration saves some calculation time.

Treating $\delta\Phi_i$ with the implicit integration may be necessary when other dynamic effects on the torsion are simulated. For example torsional friction would need a fully dynamic simulation of the twist angles. But these effects play no role in the simulation of the thread in MicroSim.

4.1.4. Spatial hashing

Section 3.5, 3.6 and 3.7 describe how contacts between the thread and the mesh and the forceps and itself are detected and handled. What is not described in these sections is which segments and vertices of the thread are tested against other objects or each other. One could e. g. test every segment of the thread against every other segment. This “Brute force” approach has the disadvantage that its time consumption grows quadratically in the number of segments and soon becomes unfeasible.

There are several approaches to overcome this problem. All approaches known to the author subdivide space in some manner. This allows avoiding collision tests for segments or vertices that are to greatly separated to be able to collide. As it is not a subject of this thesis, the various *spatial subdivision* (SSD) algorithms will not be discussed here.

For collision detections in MicroSim, the spatial hashing from Teschner et al. [111] has been applied. It divides the space into equal sized cubes. For every node or segment it tests with which cubes they overlap. Collision detection is done only for segments that have a common cube with which they both overlap. Since dividing the whole relevant space would result in a huge number of cubes, storing the contents of the cubes in an array with size equal to the number of cubes would require more memory than is feasible. The content of the cubes is therefore stored in a hash map. The hash function is based on the integer position of the cube in the grid of all cubes.

The side length of the cubes has been chosen empirically by testing for which side length the least collision tests occur. A side length which corresponds to three times the longest distance of a line within one segment has been detected to be optimal. The longest distance includes the spheres of the adjacent vertices and is therefore $2 * r + l_0$ where r is the radius of the thread and l_0 the rest length of the segment.

The number of entries in the hash table has been set by testing for optimal executing time. The best results were achieved with 2000 entries.

4.1.5. Structure of the simulation main loop

The thread is simulated with a time step of 5ms, doing 6 time steps per displayed frame. This yields a total frame time of 30ms and a reactive frame rate of 33 frames per second. The simulation of the tissue is updated twice as often as the thread and has therefore a time step of 2.5ms.

There are a few things to consider when doing the various steps for the thread simulation. First, as a side effect of the implicit integration, the forces acting on the thread are not known until the thread has been integrated. In other words, the same forces (in the opposite direction) would have to be applied to colliding objects. This cannot be done before the integration of the thread. Second the continuous collision detection potentially changes the position of individual vertices. Applying contact forces and detecting contacts of the thread with itself and other objects requires the positions of the vertices not to change until the integration. Therefore continuous collision detection is done before contact detection.

When one minds these conditions, the order of the steps in the simulation main loop is almost canonical. Here is a list of the steps with references to the sections of the work describing the individual steps in more detail.

1. Update forceps position from input system (see the work of Schuppe [96]).
2. Continuous collision detection of thread with forceps, mesh and itself. In this step the position of the thread vertices are potentially changed (section 3.8).
3. Collision detection of thread with itself (section 3.7.1).
4. Collision detection of thread with forceps (section 3.6.1).
5. Grabbing of thread with forceps (section 3.6.2).
6. Collision detection of thread with mesh (section 3.5.1).
7. Update sutures (section 3.5.2).

8. Apply all forces from collisions to thread.
9. Solve self-contacts of the thread (section 3.7.2).
10. Integrate thread position (section 3.2.2).
11. Apply forces to mesh.
12. Integrate mesh (twice, see the work of Sismanidis [102]).

4.2. Tests

4.2.1. Metrics

Floating point values (the values with which real values are represented in computers) do not have any metrical unit. But in case of physical simulation they often represent real values with metrical units. To give them meaningful values, one has to know how to convert the floating point values to get the corresponding real values.

For the simulation in this thesis, all values can be broken down to combinations of length, time and mass. In the simulation, length is measured in millimeters (mm), so a floating point of 1.0 corresponds to $1mm$. Mass is measured in grams (g). Time is a little more tedious. The simulation assumes that a simulation step has a time length of 1.0 (as a floating point value). This 1.0 corresponds to the duration of a simulation step, which is 5 milliseconds. So time is measured in units of "$5ms$".

Conversions for other units have to be represented in terms of grams, millimeters and 5 milliseconds. For example a Newton (the unit for force) is equal to

$$1N = 1kg\frac{1m}{1s^2} = 10^3 g\frac{10^3 mm}{10^6 ms^2} = 1g\frac{1mm}{1/25(5ms)^2} = 25g\frac{mm}{(5ms)^2}.$$

Therefor a force of $1N$ would be represented by 25.0 in the computer.

In the following, three test scenarios will be described. Table 4.1 lists the values used for the thread parameters in these tests. It is important to note, that these values have not been set based on material parameters but rather, by visual observing which values gave the most plausible results.

The fact that the stiffness constants k^S, k^B and k^T are directly related to distinct observable behaviors (stretching, bending and twisting) has been beneficial. Tweaking these variables by observation is easy. For example, when one observes that the thread does not resist bending enough, one knows that k^B has to be increased.

4.2.2. Test 1: Different types of knots

When binding together two ropes or threads, a common knot is the **reef knot** or **square knot**. It is performed by tying a left-handed overhand knot and then a

Scenario:	**Knot test**	**Torsion test**	**MicroSim**
mobility $[kg/s]$	7.47	37.35	37.35
mobility (float)	37.35	186.73	186.73
$k^S[kgm^2/s^2]$	$1.25 \cdot 10^8$	$1.25 \cdot 10^6$	$1.25 \cdot 10^8$
k^S (float)	5000	50	5000
$k^B[kgm^2/s^2]$	$1.25 \cdot 10^4$	$1.25 \cdot 10^3$	$2.5 \cdot 10^4$
k^B (float)	0.5	0.05	1.0
$k^T[kgm^2/s^2]$	$2.5 \cdot 10^4$	$2.5 \cdot 10^4$	$2.5 \cdot 10^4$
k^T (float)	1.0	1.0	1.0
thread radius $[m]$	$5 \cdot 10^{-4}$	10^{-4}	10^{-4}
thread radius (float)	0.5	0.1	0.1
$l_0[m]$	$5 \cdot 10^{-4}$	10^{-4}	10^{-4}
l_0 (float)	0.5	0.1	0.1
Coeff. of Friction	varying	0.95	0.95
number of vertices	150	200	600

Table 4.1.: Values of the parameters for the different tests

right-handed overhand knot (or vice versa, see figure 4.4(a)). A overhand knot is simply a rotation of the two knots around the longitudinal axis.

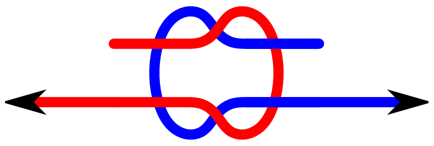
One can vary the reef knot in different ways. If two overhand knots with the same hand-side are performed the so called **granny knot** results (see figure 4.4(b)). Although it is similar to the reef knot, it is inferior in the sense that it can hold slightly less load.

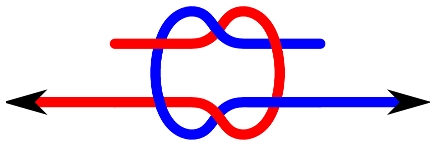
Another variation is the **thief knot** which resembles the reef knot, except that the free ends are on opposite sides (see figure 4.4(c)). The **grief knot** has the same relation to the granny knot as the thief knot to the reef knot as it is pulled by using opposite sides of a granny knot as free sides (see figure 4.4(d)). It is highly insecure.

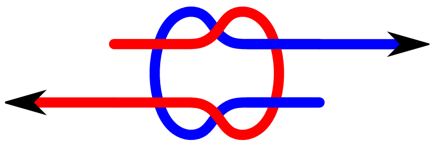
In summary, the reef knot is the most secure of the knots presented here. The granny knot is less secure while the thief and grief knot are the most insecure knots. It would be nice if the simulation would somehow resemble this relation.

A thread with the physical parameters given in table 4.1 column 1 is created forming one of the knots. A force of $1N$ is applied in both directions. The coefficient of friction μ is reduced (starting from a value of 0.95) in steps of 0.05 until the knot dissolves.

To resemble reality, the coefficient of friction for which the reef knot dissolves should be the lowest, while the grief knot should dissolve with highest coefficient of friction. The results are given as four videos in the supplemental material or on


(a) Reef knot


(b) Granny knot


(c) Thief knot

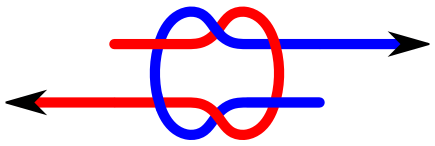

(d) Grief knot

Figure 4.4.: Variants of the reef knot

YouTube [1]. For this simulation the segments have been colored alternating red and green to better visualize the individual segments.

The reef knot tightens itself at several steps where the coefficient of friction is reduced. At $\mu = 0.5$ it is slightly unstable while at $\mu = 0.45$ the knot starts to dissolve. The granny knot seems to dissolve at $\mu = 0.7$, but gets stable again. Similar to the reef knot it gets unstable at $\mu = 0.5$ and dissolves at $\mu = 0.45$. At this point it dissolves much faster than the reef knot. The thief knot starts dissolving much earlier. At $\mu = 0.7$ it seems to dissolve very slowly, while at $\mu = 0.65$ the dissolving is clear. The grief knot starts dissolving with $\mu = 0.55$.

This result partly resembles reality. The reef knot is the most secure one, but only very slightly compared to the granny knot. When one performs tests with real threads, these results do not seem to far off. The secureness of reef and granny knot does not differ that much. The result that the thief knot dissolves so fast is nice, but is reduced by the fact that the grief knot is almost as secure as reef and granny knots, which does not correspond to reality.

The question of when a knot dissolves is not only guided by physical parameters but also by how good the NNCG converges, which is a very unphysical property. When the NNCG converges badly, the movement of the thread gets unsteady, allowing a knot to dissolve faster. Unfortunately the convergence of the NNCG is limited by computing resources.

[1] Reef-knot: `http://youtu.be/i9YlzAAa56s`
Granny-knot: `http://youtu.be/Ok9RunnHkSU`
Thief-knot: `http://youtu.be/9KnAUMbIabI`
Grief-knot: `http://youtu.be/ERS26Y1IkG4`

4.2.3. Test 2: Torsion

When holding one end of a thread, while twisting the other, torsional energy builds up. The thread can release the torsional energy by moving towards a configuration where the twist is reduced. When the position of the thread's end points is fixed, this configuration has higher bending energy. So the thread moves towards a configuration where bending and torsional energies are balanced.

This produces configurations with visually recognizable qualities. See figure 4.5.

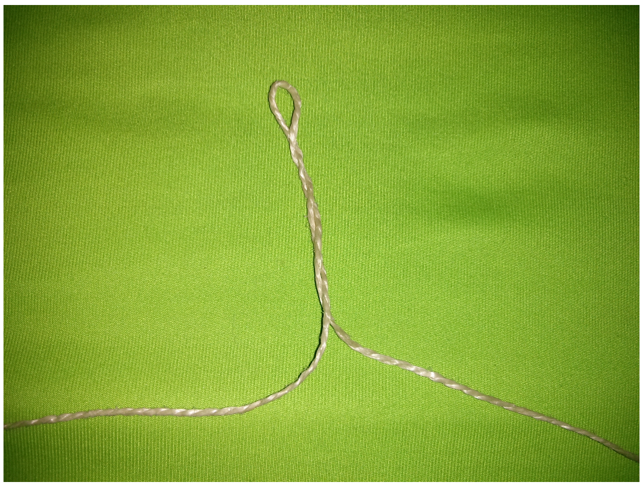


Figure 4.5.: A twisted thread

This scenario has been reproduced in a simulation. A thread with the values found in the third column of table 4.1 has been created. The position of the ends have been fixed while the material frames of the outer most segments have been rotated around the corresponding tangent in opposite direction. The total twist changes with 0.8 radiant per second.

Again, segments have been colored alternating red and green to better visualize the individual segments. A video of the simulation is given in supplemental material or on YouTube[2]. A screen shot of the resulting configuration is shown in figure 4.6.

Comparing figure 4.5 and 4.6 one can clearly see that simulation can reproduce the expected real behavior.

4.2.4. Test 3: Anastomosis in MicroSim

An anastomosis is performed partly in MicroSim. In the initial setup, the thread is pierced through the ends of the vessels so that it can be knotted above where the

[2]`http://youtu.be/aHjnR4Uahp4`

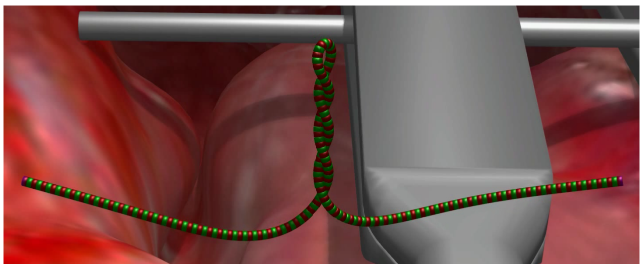


Figure 4.6.: Simulation of the twisted thread.

vessels meet (see figure 4.7(a)).

Using the marker-tracking controlled forceps, a reef knot is performed pulling the vessels together. Figure 4.7(b) shows the situation after the first overhand knot has been performed. The final result of the anastomosis is shown in figure 4.7(c).

A video of the complete anastomosis is given in the supplemental material or on YouTube[3]. It demonstrates the capabilities of MicroSim. But it also shows its limits. The knot in the ends seems a little large. And indeed, a real surgical thread for this anastomosis has a diameter of $0.02mm$, while the simulated threads diameter is ten times as large ($0.2mm$).

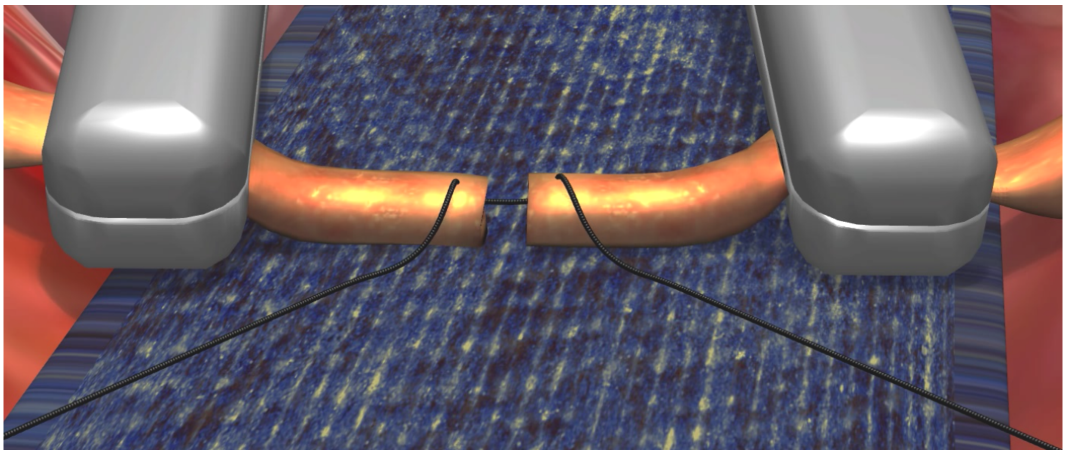
Unfortunately there are limits to how small the thread may be. In particular the NNCG solver does converge much worse with a smaller thread radius resulting in unstable knots. This can be counteracted by reducing the mobility, but the thread already seems to move rather slowly. In addition, tunneling of thread segments occur much more frequently with a smaller radius. Also, when the thread needs to keep its bending agility, the segment length must be reduced, requiring a higher number of vertices for maintaining the threads length. This especially increases the computational burden of the collision detection.

Note that at one point the simulation seems to stop for a short moment. This is an artifact of the video recording software and does not happen when no screen recording is active. Presumably at the moment the simulations seems to stop when the recording software is writing to disc or demanding memory from the operation system.

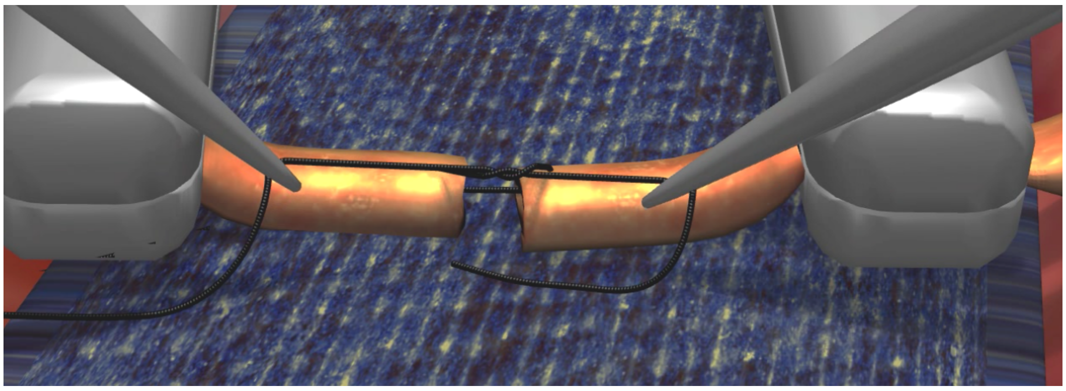
4.3. Performance

As the thread simulation in this thesis is supposed to work in real time applications, it is important to analyze its computing time requirements. Time measurements have been done for the tests listed above in which a special attention has been given

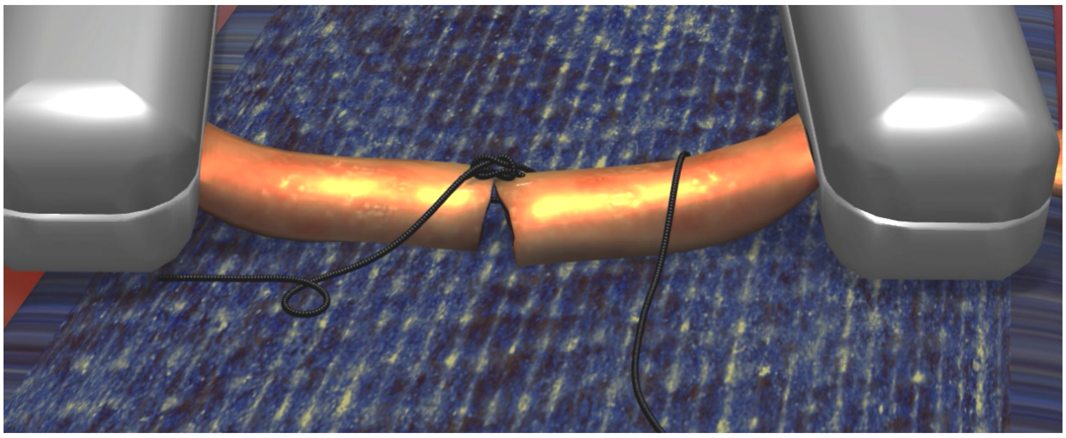
[3]`http://youtu.be/ZWkQga7RIHc`



(a) Initial setup



(b) Performing the first overhand knot



(c) Final result

Figure 4.7.: Performing an anastomosis in MicroSim

to the time consumption of the constraint solver.

The time measurements have not been done in the same execution of the program as the video recording. This is because the video recording interferes with the performance as it requires CPU time and memory. For the MicroSim test, the input has been recorded and replayed for the measurement.

4.3.1. Time measurements in the tests

For all 3 tests discussed above, the amount of time various aspects of the simulation consume in a frame have been recorded. Since there are 6 simulation iterations per frame, these timings are summed over 6 invocations. The results are shown in figure 4.8, 4.9 and 4.10. For the knot test, only the test with the reef-knot has been taken exemplary. The keys have the following meanings:

- **Total frame:** The total duration of the frame. To stay within 30 frames per second, this should stay below $33ms$.
- **Thread integration:** Integration of the thread.
- **Internal forces:** Calculation of all internal forces of the thread.
- **Constraint solver:** Resolution of the contact constraints including building of the coupling matrix.
- **CD (betw. threads):** Collision detection of the thread with itself, or between different threads. This includes building of the hash map for segments and continuous collision detection.
- **Contacts w/ forceps & tissue:** Contact detection and resolution (including continuous collision detection) between the thread and the tissue and forceps. This includes building of the hash map for vertices.
- **Update Mesh:** Integration of the mesh, which represents the tissue. Interaction (collision detection and resolution) between tissue and forceps.

In the plot for the torsion test (figure 4.8) and in the plot for the MicroSim test (figure 4.10 and 4.11), one may notice that the time consumed by thread integration doubles when the constraint solver uses time. This is because when contact constraints must be solved, the thread needs to be integrated twice. The first integration is required to predict where colliding segments would move without contact forces (see section 3.7).

For the **torsion test** (see figure 4.8) the total frame time stays below $33ms$ but till the end of the test. As the number of contacts increase, the time consumed

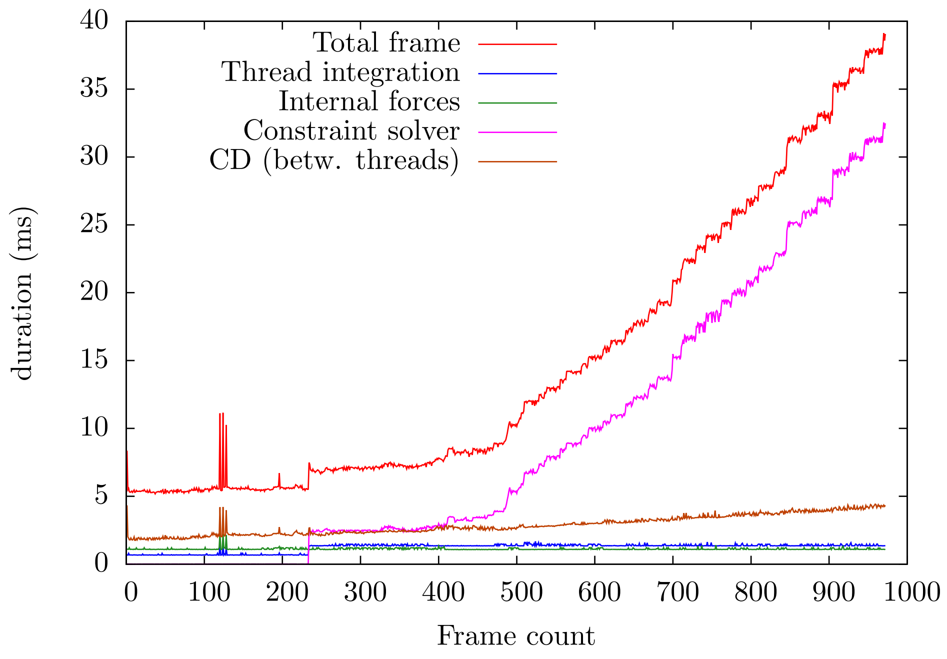


Figure 4.8.: Performance analysis for the Torsion test

by the constraint solver increases and with it the total frame time. In the end the constraint solver is the clear bottleneck, as it consumes most of the computation time.

The collision detection and resolution between the threads slowly increases because the spatial hashing outputs more potential collisions and more collisions tests have to be done. Never the less, it stays below $5ms$. The torsion test uses the least number of vertices for the thread. All factors but the constrain solver in sum stay below $6ms$.

For the **knot test** (see figure 4.9) the contribution of the thread integration also stays below $4ms$, which is less than an eight of the total allowed frame time. Most of time, integration takes about $2.5ms$ in total. For about the first 350 frames the total frame time is much higher than for the rest of the test. At its peaks it is above $70ms$. From the graph it is clear that the constraint solver is mainly responsible. Because in the beginning of the test, where the knot falls into place, there are a lot of self contacts of the thread. This is not captured by the video.

For the MicroSim test (see figure 4.10), the real time capabilities of the simulation reach their limits. While in the absence of any knots the requirement of a maximum frame time of $33ms$ is barely met, with knots the frame time goes up as high

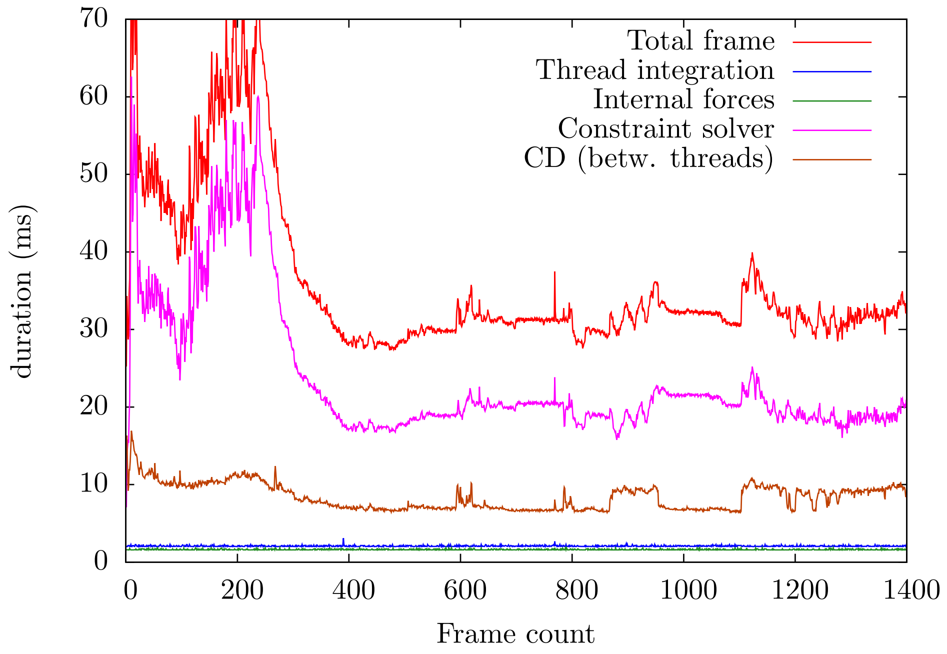


Figure 4.9.: Performance analysis for the Knot test

as almost $70ms$ (which corresponds to about 14 frames per second). For a short moment around frame count 1200 it reaches almost $100ms$. A frame time of $100ms$ corresponds to a frame rate of around 10 frames per second. It is still possible to work with these low frame rates, but the simulation feels sluggish.

Figure 4.11 shows the first 200 frames of the MicroSim test in more detail. Integrating the thread with 600 vertices takes no more than $5ms$, even when the integration has to be done twice (for a short moment before frame 150).

The constraint solver is, as expected, the biggest variance in the execution time. When it goes up, the whole frame rate drops. Other important contributors to the frame time are the mesh forceps interaction (around $15ms$), the collision detection between the thread and itself (varying between $5ms$ and $9ms$) and the update of the mesh (around $10ms$).

4.3.2. Constraint solver

The last section clearly shows that when knots are required the dominant performance bottleneck is the constraint solver. For a real time application such as MicroSim it is important that the frame rate stays above a certain rate at all times. It is not acceptable when the frame rate drops significantly, even if it happens only

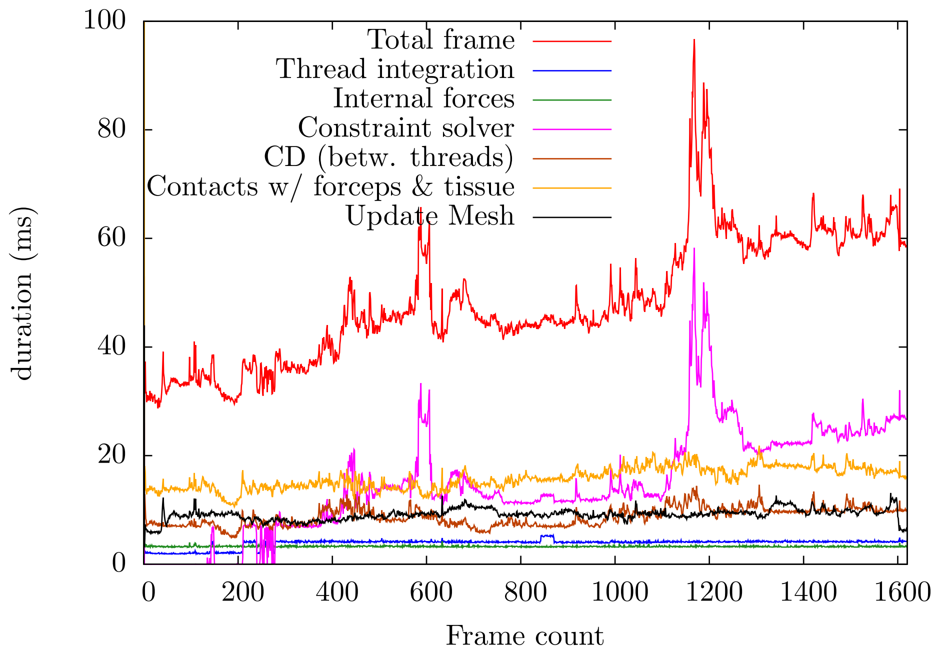


Figure 4.10.: Performance analysis for the MicroSim test

in specific situations. Therefore, the worst case scenario is the most important to improve.

When there are many self-contacts of the thread, the constraint solver is the clear bottleneck. The constraint solver consist of two main parts: Building of the coupling matrix and solving for the contact forces. As stated in section 3.7.2, building of the coupling matrix has a complexity of $O(M^2 + N)$ were M is the number of contacts and N the number of vertices in the thread. Since the number of iterations in the constraint solver is set to a fixed number, solving for the contact forces has a complexity of $O(M^2)$. In all the tests the constraint solver performs 10 iterations.

The timings for building the coupling matrix, solving for contact forces and the total time of the constraint solver have been measured for the tests described above. For the knot test, they have only been measured for the reef knot. Figure 4.12 and 4.13 show the time measurements for building the coupling matrix and solving for the constraint forces respectively.

The quadratic dependence on the number of constraints is visible for all three tests in figure 4.12 as well as in figure 4.13. The impact of the number of vertices in the thread can be seen at the left border of figure 4.12. Since there are very few contacts, the time consumption due to the quadratic dependence on the number of

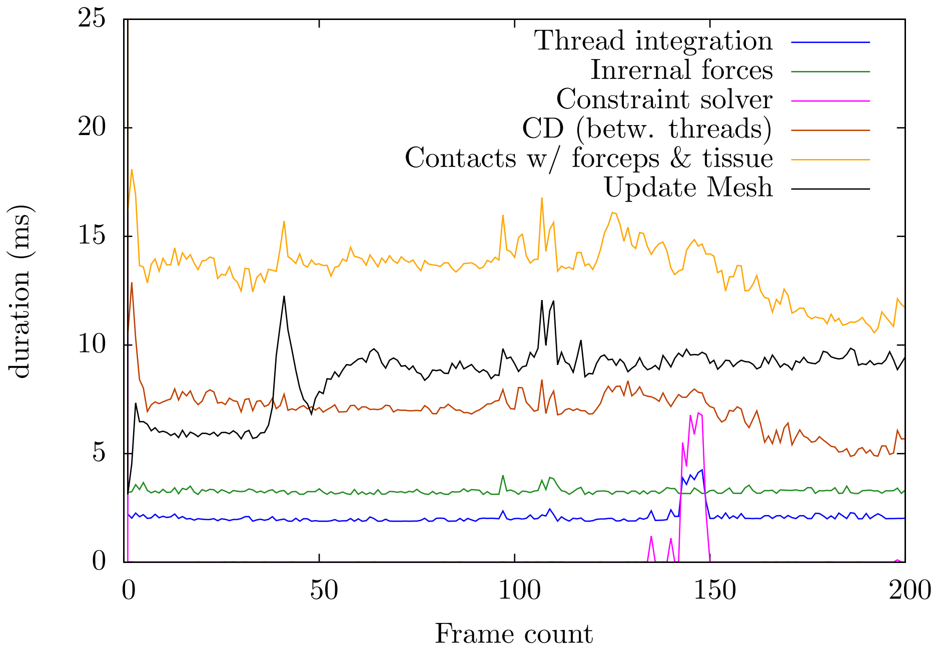


Figure 4.11.: Performance analysis, only first 200 frames.

contacts is low. The linear time dependence on the number of vertices on the other hand is unchanged. In the MicroSim test, the thread is made of significantly more vertices than in the knot or torsion test. In accordance with this the measurements for the MicroSim tests start at about $0.8ms$ while for the knot and torsion test they are at about $0.2ms$ at the left border of figure 4.12.

Figure 4.12 and 4.13 also reveal that for all tests, the time consumed by building the coupling matrix is much higher than the time consumed by finding the contact forces. One has to be aware that this changes when one increases the number of iterations for the NNCG solver. For the tests, the number of iterations was set to 10, which maintains the real time capabilities of the simulation. For more accurate knot simulation, a higher number of iterations is called for. Iterations of 100 or even more would be desirable. The time consumption for finding the contact forces is linear in the number of iterations. So this change would multiple the time consumption by 10, moving it in the same scale as the time consumption for building the coupling matrix.

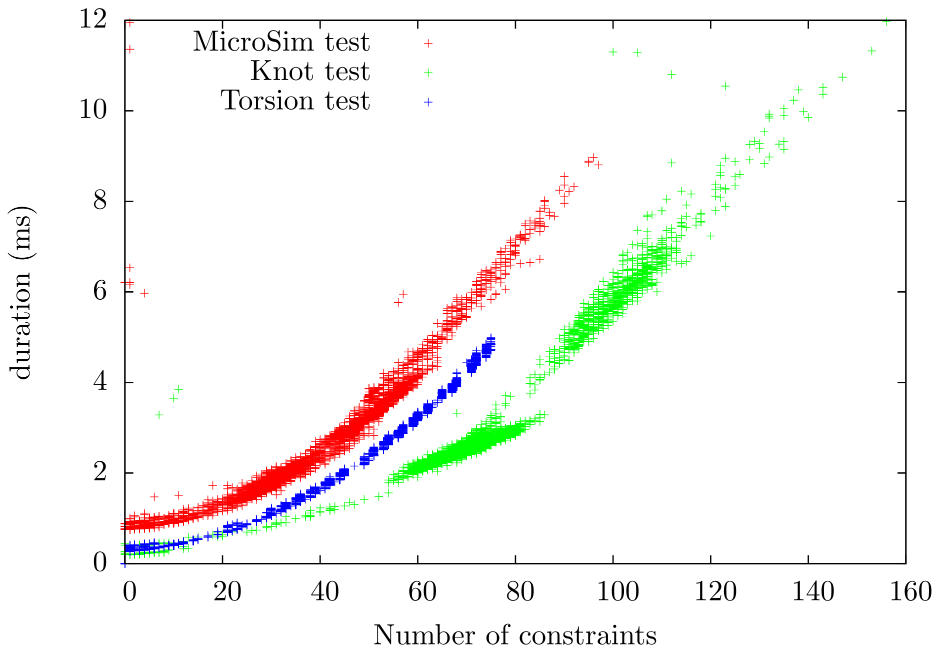


Figure 4.12.: Timings for building the couplings matrix

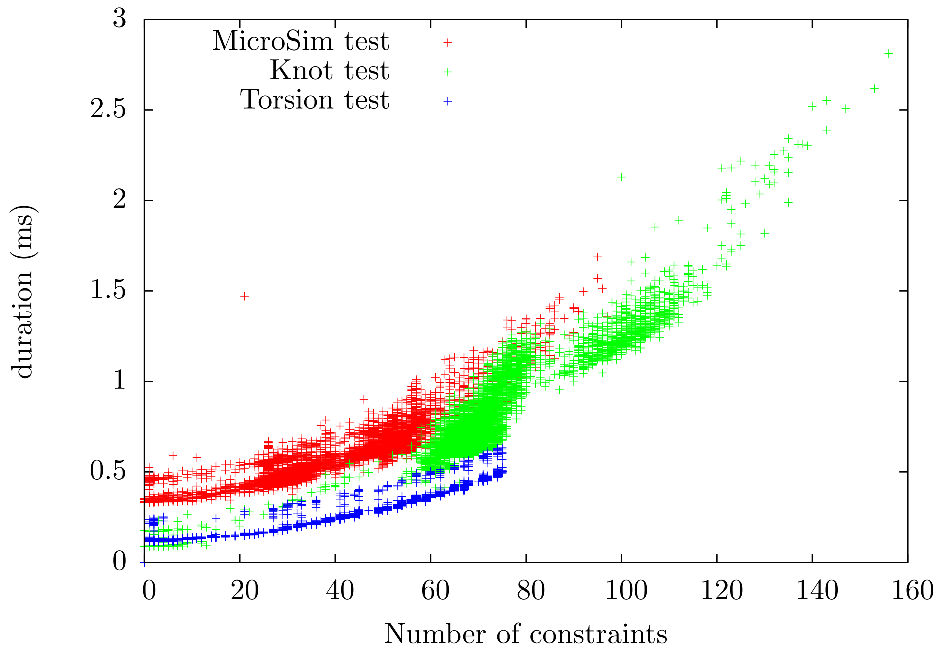


Figure 4.13.: Timings for solving for contact forces

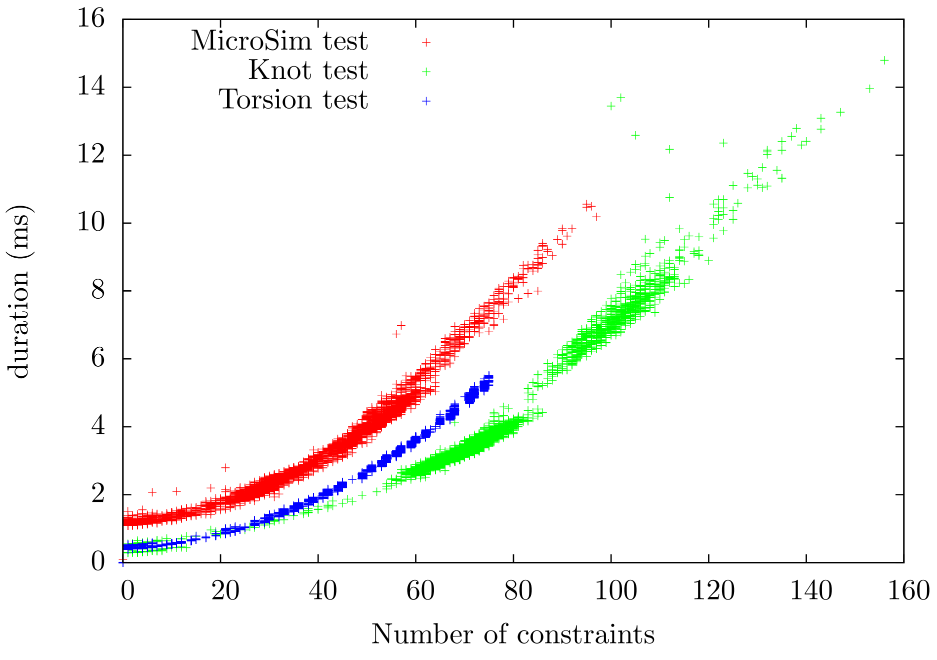


Figure 4.14.: Timings for the NNCG constraint solver

5. Discussion and Outlook

In this thesis, the simulation of thread, integrated with an implicit integration scheme, has been developed. The thread is modeled with a mass spring model. The challenge of simulating threads with high stiffness has been addressed with implicit integration, which allows the thread to have practically unlimited stiffness. The simulation runs with a very large time step ($5ms$, for comparison [104] uses a time step of $0.1ms$) making it real time capable.

The large time step has also the effect that the thread potentially moves by a multiple of its diameter within a single simulation step. To prevent tunneling effects, continuous collision detections between the thread and itself as well as all other objects has been applied.

The implicit integration requires that all applied forces are given, including their derivative with respect to the degrees of freedom in the thread. Such forces have been calculated for torsion, bending and stretching of the thread. The physical parameters of the thread have been chosen to directly control these qualities. This allows the adjustment of the thread based on visual observation of its behavior. For example, if it is required to stronger resist bending, there is a parameter directly controlling this feature.

For collision resolution with other objects (the tissue and the forceps), collision responses suited for the implicit integration have been developed. For collisions of the thread with itself, there are higher demands on the quality of the collision response. To simulate knots in a realistic way, accurate contact forces have been calculated. Because of the implicit integration, all contacts interact. The strength of this interaction has been calculated in a coupling matrix allowing the contact problem to be formulated as a nonlinear complementary problem. Inspired by rigid body contact problems, a nonsmooth conjugate gradient method (see also [101]) has been applied.

One may note that instead of calculating the interactions between the contacts, one could add the contact constraints to the linear system that is solved by the implicit integration. But this would add entries outside the bands of the matrix of the linear system and transform the problem of integrating into a complementary problem. Solving this system would require too much computation time for an interactive application. By calculating the interaction of the constraints and solving the constraint in a separate step, the problem has been separated. The elements outside the bands and the complementary equations are solved in a separate step.

The solution is the same as the much larger complementary problem. But the difficult complementary problem has now been reduced to size M instead of N, where M is the number of contacts and N the number of degrees of freedom of the thread. The remaining problem is the banded matrix of the implicit integrator, which can be solved in linear time.

The simulation has been integrated into the microsurgical training simulator MicroSim and it has been demonstrated that a virtual anastomosis can be performed. In addition it has been shown that knots can be simulated and that torsion and bending forces interact in the expected ways. The behavior of the thread can be adjusted with a number of parameters which are directly connected to visually observable behaviors. This allows for tweaking the thread and for adjusting observable features directly.

When other time demanding calculations have to be done in addition to the thread simulation (such as simulation of tissue), the real time capabilities have their limits. The virtual anastomosis cannot be run with the full 30 frames per seconds, but reaches frame rates as low as 10 frames per seconds. Shifting this limit would be very desirable. The most limiting factors are the collision detection and the constraint solver. The integration of the thread itself is not very time demanding.

Other works have extended the real time capabilities of different thread simulations by making the length of segments dynamic (see e. g. [104]). When the bending between segments is higher than a threshold, segments are divided decreasing the bending. When the bending goes below a threshold, segments are combined. This especially reduces the computational demands of the integration, which is not a limiting factor here as has also been discusssed in section 3.1.2. Since inside of knots the thread would be maximally divided, the constraint solver would not benefit. The collision detection could benefit from fewer elements, but the spatial subdivision scheme used in this thesis (the hash map from [111]) cannot deal with elements of different sizes efficiently. A different spatial subdivision scheme, such as Octrees [22], could be applied. Then it could also be designed to take advantage of aligned segments.

A large time step also has its disadvantages. If an assumption (such as a calculated force) is not correct for the full time step, the error builds up more with a large time step. A slightly smaller time step would be nice but is difficult because of the computational demands. Collision detection would probably not be necessary in every integration step. For example in every second step the collision information from the last step could be updated, without detecting new collision.

A time intensive requirement that is needed in every step is the constraint solver. It could benefit from a lower time step due to warm starting and require less iterations. Still, the building of the coupling matrix would be required in every step and it would not benefit from warm starting.

So in general, a speedup of the constraint solver would be very desirable. If more iterations could be performed, the radius of the thread could be decreased to a realistic value and its mobility increased without hurting the convergence of the solver. Probably a parallelization of the algorithm would yield the required performance gain. The main part is a large matrix calculation, which could be parallelized to a certain level. Another important aspect is calculating the coupling matrix. The main computation burden is finding the $H_{i \to j}$ which could also be done in parallel.

The virtual anastomosis in MicroSim is far from simulating the complete procedure. It starts at a point where the thread has already been stitched into the blood vessels and stops when the first stitch has been knotted. To simulate the complete procedure including piercing the blood vessel with a needle, an interaction between needle and thread has to be developed. The thread needs to be attached onto the needle and follow its movements. This could e. g. be done with the same technique used for grabbing the thread with the forceps. When more stitches should be performed, the time demands of the constraint solver become even more problematic than they are now because it would have to solve several knots at once. When there are many knots, the coupling matrix could be split into strongly connected components where small entries are thresholded. Then each component could be solved on its own. Since the solver complexity is quadratic in the number of contacts, in the presence of many knots a big gain would be attained. Also knots could be detected by their topology and when they are known to hold, they could be transformed into rigid objects that are connected to the thread. Sismanidis [102] developed other training modules for microsurgical procedures not involving the thread.

Testing when a reef, granny, thief, or grief knot dissolves shows that knots can be simulated realistically, at least to a certain extent. The realism fails when the constraint solver does not converge fast enough. So this is again a problem of the time consumption of the constraint solver. Still, a knot holding together the blood vessels can be performed while the frame rate of the simulation stays high enough for interactivity. However, a very loose knot, such as a simple loop, would not hold under any conditions. Just as a tutor teaching medical students would check if the students performed the correct knot suture, it would be desirable to have an automatic classification of the knot a trainee performed in the training simulator. The trainee could get feedback if he performed the correct knot or his test score could be adjusted accordingly. Such a classification could be done by a topological analysis of the knot.

A. Example for implicit integration: Harmonic Oscillator

Here an example of the unconditional stability of implicit integration is given.

The harmonic oscillator is a physical system described by the potential $U(\vec{x}) = \frac{1}{2}k\vec{x}^2$ where $k > 0$. It results in the force

$$\vec{F}(\vec{x}) = \nabla U(\vec{x}) = -k\vec{x}$$

and is the potential of a spring with vanishing rest length. Whenever springs are used in the physical model, the harmonic oscillator or a variant of it is applied. Here it is shown that the implicit integration scheme from section 3.2.2 with the semi implicit Euler integrator is always stable when applied to the harmonic oscillator potential. This is done by showing that the energy never increases in an integration step.

Assume an object with mass m, position $\vec{x}(t)$ and velocity $\vec{v}(t)$ is in the potential of the harmonic oscillator.

Let t_0 be the time before the integration step. Δt is the time step and $t_1 := t_0 + \Delta t$ the time after the integration step. The starting state of the system is defined by $\vec{x}(t_0)$ and $\vec{v}(t_0)$. To create a state vector with uniform units, $\vec{w}(t) := \vec{v}(t) \cdot \Delta t$ is used. The energy $E(t)$ of the system is

$$E(t) = \frac{1}{2}\left(k \cdot \vec{x}(t)^2 + M\vec{w}(t)^2\right)$$

where $M := m/\Delta t^2$.

To shorten notation, so that the time argument does not always have to be written down explicitly, $\vec{w}(t_i), \vec{x}(t_i)$ and $E(t_i)$ are written as $\vec{w}_i, \vec{x}_i$ and E_i respectively.

The integration updates position and velocity according to:

$$\begin{aligned} \vec{w}_1 = \vec{w}_0 - \frac{k}{M}\vec{x}_1 \quad &, \quad \vec{x}_1 = \vec{x}_0 + \vec{w}_1 \\ \Rightarrow \vec{x}_1 = \alpha\vec{x}_0 + \alpha\vec{w}_0 \quad &, \quad \vec{w}_1 = (1-\beta)\vec{w}_0 - \beta\vec{x}_0 \end{aligned}$$

where

$$\alpha := \frac{1}{1+\frac{k}{M}} \quad , \quad \beta := \frac{k}{M}\alpha = \frac{1}{1+\frac{M}{k}}$$

Stability will be proven by proving that the energy of the system decreases. For this it will be shown that $2E_0 - 2E_1 > 0$.

$$\begin{aligned} 2E_0 - 2E_1 &= k\vec{x}_0^2 + M\vec{w}_0^2 - k\vec{x}_1^2 - M\vec{w}_1^2 \\ &= a\vec{x}_0^2 + b\vec{w}_0^2 - 2c\vec{x}_0\vec{w}_0 \end{aligned}$$

where

$$\begin{aligned} a &:= k - k\alpha^2 - M\beta^2 \\ b &:= M - k\alpha^2 - M\left(1-\beta\right)^2 \\ c &:= k\alpha^2 - M\left(1-\beta\right)\beta \end{aligned}$$

This can be expressed in the quadratic form:

$$\begin{pmatrix}\vec{x}_0\\ \vec{w}_0\end{pmatrix} A \begin{pmatrix}\vec{x}_0\\ \vec{w}_0\end{pmatrix} \quad , \quad A := \begin{pmatrix} a\mathbb{1} & -c\mathbb{1} \\ -c\mathbb{1} & b\mathbb{1}\end{pmatrix}$$

where $\mathbb{1}$ is the unit matrix. The quadratic form is always positive iff the eigenvalues of A are all positive. A has the same eigenvalues as the 2×2 matrix $A' := \begin{pmatrix} a & -c \\ -c & b\end{pmatrix}$. The sum of A' eigenvalues is given by the trace of A' while there product is given by A's determinant $\frac{1}{ab-c^2}$. If both are positive than A's eigenvalues are also positive, proving the claim.

Because k and M are positive, it must be that $0 < \alpha < 1$.

$$\begin{aligned} & & 1 &> \alpha \\ \Rightarrow & & 1 &> 2\alpha - \alpha = 2\left(1+\frac{k}{M}\right)\alpha^2 - \alpha \\ \Rightarrow & & 1 - 2\alpha^2 &> 2\frac{M}{k}\left[\left(\frac{k}{M}\alpha\right)^2 - \frac{k}{M}\alpha\right] \\ \Rightarrow & & k\left(1-\alpha^2\right) &> M\left(2\beta^2 - 2\beta\right) \\ \Rightarrow & & 0 &< k\left(1-2\alpha^2\right) + M\left(2\beta - 2\beta^2\right) = a + b \end{aligned}$$

Showing that the determinant is positive is done by showing, that $ab - c^2 > 0$.

$$
\begin{aligned}
& \frac{k}{M} > 0 \\
\Rightarrow \quad & 2 + 2\frac{k}{M} > 2 + \frac{k}{M} \\
\Rightarrow \quad & 2 > (2 + \frac{k}{M})\alpha = 2\alpha + \beta \\
\Rightarrow \quad & 2 - 2\alpha - \beta > 0 \\
\Rightarrow \quad & k^2\alpha\,(2 - \beta - 2\alpha) = kM\beta\,(2 - \beta - 2\alpha) = kM\left(2\beta - \beta^2 - 2\alpha\beta\right) > 0 \\
\Rightarrow \quad & ab - c^2 \\
& = kM\left[1 - (1-\beta)^2 - \alpha^2 + (1-\beta)^2\alpha^2 + \beta^2\alpha^2 + 2\alpha^2(1-\beta)\beta\right] - k^2\alpha^2 - M^2\beta^2 \\
& = kM\left(2\beta - \beta^2\right) - \alpha^2 k^2 - M^2\beta^2 = kM\left(2\beta - \beta^2 - 2\lambda\beta\right) > 0
\end{aligned}
$$

with which the statement is proven.

B. Geometric tests

In this appendix, various geometric tests, needed for collision detection, are given.

B.1. Barycentric Coordinates

Barycentric coordinates are a way to express a position within a triangle as a linear product of the triangle's corners. Let $\vec{a}$, $\vec{b}$ and $\vec{c}$ be a triangle with normal $\vec{n}$. $\vec{n}$ is defined by

$$\vec{n} := \frac{(\vec{c} - \vec{a}) \times (\vec{b} - \vec{a})}{|(\vec{c} - \vec{a}) \times (\vec{b} - \vec{a})|}$$

which also defines the direction in which $\vec{n}$ points. The area of the triangle is given by

$$A = \frac{1}{2} \Big((\vec{c} - \vec{a}) \times (\vec{b} - \vec{a}) \Big) \cdot \vec{n}.$$

Let $\vec{p}$ be a point inside the triangle. Its barycentric coordinates is a triple $(t_1, t_2, t_3) \in \mathbb{R}^3$. Each element of the triple corresponds to a corner of the triangle. When one replaces one of the corners of the triangle with $\vec{p}$ and calculates the fraction this triangle has of the whole triangle area, one gets the corresponding barycentric coordinate (see figure B.1).

$$t_1 := \frac{1}{2A} \Big((\vec{c} - \vec{p}) \times (\vec{b} - \vec{p}) \Big) \cdot \vec{n} \tag{B.1}$$

$$t_2 := \frac{1}{2A} \left((\vec{c} - \vec{a}) \times (\vec{p} - \vec{a}) \right) \cdot \vec{n} \tag{B.2}$$

$$t_3 := \frac{1}{2A} \Big((\vec{p} - \vec{a}) \times (\vec{b} - \vec{a}) \Big) \cdot \vec{n} \tag{B.3}$$

$$\tag{B.4}$$

The sum of the coordinates represents the fraction for which the whole triangle is of itself. Therefore

$$t_1 + t_2 + t_3 = 1.$$

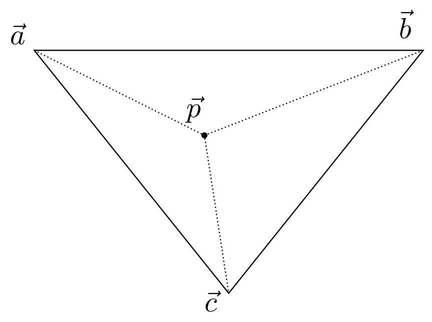


Figure B.1.: Point inside a triangle forming 3 triangles whose areas correspond to the barycentric coordinates.

Another important property is, that:

$$\vec{p} = t_1\vec{a} + t_2\vec{b} + t_3\vec{c}.$$

This can be shown by inserting $t_1\vec{a}+t_2\vec{b}+t_3\vec{c}$ into (B.1) - (B.3) and showing that this point has the same barycentric coordinates as $\vec{p}$. Here this will be done exemplary for (B.3):

$$\begin{aligned}
t_3' &:= \frac{1}{2A}\Big((t_1\vec{a} + t_2\vec{b} + t_3\vec{c} - \vec{a}) \times (\vec{b} - \vec{a})\Big) \cdot \vec{n} \\
&= \frac{1}{2A}\Big(t_3(\vec{c} - \vec{a}) \times (\vec{b} - \vec{a}) + (t_1\vec{a} + t_2\vec{b} + t_3\vec{c} - (1 - t_3)\vec{a}) \times (\vec{b} - \vec{a})\Big) \cdot \vec{n} \\
&= \frac{1}{2A}\Big(t_3(\vec{c} - \vec{a}) \times (\vec{b} - \vec{a})\Big) \cdot \vec{n} + \Big((t_1\vec{a} + t_2\vec{b} - (t_1 + t_2)\vec{a}) \times (\vec{b} - \vec{a})\Big) \cdot \vec{n} \\
&= \frac{1}{2A}t_3 2A + \frac{1}{2A}\Big(t_2(\vec{b} - \vec{a}) \times (\vec{b} - \vec{a})\Big) \cdot \vec{n} \\
&= t_3
\end{aligned}$$

Equations (B.1) - (B.3) allow for the calculation of the barycentric coordinates. A more efficient algorithm is given in [37, chapter 3.4].

Note that (B.1) - (B.3) use signed areas to calculate t_1, t_2 and t_3. This means that the area fractions can get negative. This only happens when $\vec{p}$ is outside of the triangle. So a test for whether a point in a plane is inside a given triangle is calculating the barycentric coordinates and testing of they are all bigger or equal to 0.

B.2. Intersection between a plane and a line

Let the line be described by $\vec{K}(s) := \vec{a} + \vec{b}s$. The plane is given by its normal $\vec{n}$ and a point $\vec{p}$ it contains. For a given s the distance between $K(s)$ and the plane is:

$$d = \vec{n} \cdot \left(\vec{K}(s) - \vec{p}\right) \vec{n} \cdot \left(\vec{a} + \vec{b}s - \vec{p}\right)$$

The intersection is given by the point for $d = 0$. This is the case when

$$s = \frac{\vec{n} \cdot (\vec{p} - \vec{a})}{\vec{n} \cdot \vec{b}}$$

B.3. Closest point on a line to a point

Let $\vec{p}$ be the point for which the closest point on the line $\vec{a} + \vec{b}t$ has to be found. The closest point on the line to $\vec{p}$ is the projection of P onto the line [35]. Or, in other terms, the closest point on the line is the intersection of the line with the plane with the normal $\vec{b}/\left|\vec{b}\right|$ containing $\vec{p}$.

The distance between $\vec{a}$ and the plane is $(\vec{p} - \vec{a}) \cdot \vec{b}/\left|\vec{b}\right|$. Since $\vec{b}$ is orthogonal to the plane, the intersection point is:

$$\vec{p}_0 := \vec{a} + \vec{b}\frac{(\vec{p} - \vec{a}) \cdot \vec{b}}{\left|\vec{b}\right|^2}$$

B.4. Closest point on a triangle to a line

Finding the closest point on a triangle is a matter of combining the tools presented so far.

The point is projected onto the plane the triangle lies in. It is then determined whether the projected point is inside the triangle by calculating the barycentric coordinates. If this is the case, then the projected point is the closest point. Otherwise the closest point lies on one of the borders of the triangles. In this case the closest points on the line segments making the border of the triangles are calculated. The one of these points which is closest to the test point is the desired closest point.

B.5. Closest point between two lines

Let $K(s) := \vec{a} + \vec{b}s$ and $L(t) := \vec{c} + \vec{d}t$ be the lines for which the closest distance needs to be found. The vector between the closest points is orthogonal to $\vec{d}$ and $\vec{b}$. It is therefore proportional to

$$\vec{s} := \vec{b} \times \vec{d}$$

If $|\vec{s}| = 0$ then the two lines are parallel. The closest points can be found by choosing an arbitrary point on $K(s)$ and find the closest point on $L(s)$ to it (as in section B.3).

The set of points can be reached from any point on $K(s)$ following a vector proportional to $\vec{s}$ is given by the plane with normal $\vec{s}/|\vec{s}|$ which include $\vec{a}$. The closest point on $L(t)$ is given by the intersection of this plane and $L(t)$ (see appendix B.2).

$$t = \frac{\vec{s} \cdot (\vec{c} - \vec{a})}{\vec{n} \cdot \vec{s}}$$

For a more detailed derivation as well as an implementation, look into the document from Eberly [34],

B.6. Closest point on surface of a cone to a point

Finding the closest point on the surface of a cone is described in [1]. Here a short summary is given. Let $\vec{P}$ be the point to which the closest point is required. Let further $\vec{P_0}$ denote the closest point. If $\vec{P_0}$ is on the bottom plane of the cone, it is found the same way as for the half circle at the bottom of the quarter sphere. Otherwise it is on the coat of the cone. If one would extend a vector from $\vec{P}$ through $\vec{P_0}$ it would hit the center line of the cone. $\vec{P_0}$ is therefore in the plane formed by $\vec{P}$ and the end points of the center line. Intersecting this plane with the coat of the plane results in 2 lines, $\vec{P_0}$ is on the line closer to $\vec{P}$ (see figure B.2). Let $\vec{C_0}$ be the closest point in the center line to $\vec{P}$ and $\vec{C_1}$, $\vec{C_2}$ the endpoints of the center line. The line on which $\vec{P_0}$ must be is between the following points:

$$\vec{D_1} := \vec{C_1} + \frac{\vec{P} - \vec{C_0}}{\left|\vec{P} - \vec{C_0}\right|} r_1$$

$$\vec{D_2} := \vec{C_2} + \frac{\vec{P} - \vec{C_0}}{\left|\vec{P} - \vec{C_0}\right|} r_2$$

$\vec{P_0}$ is found by finding the closest point on the line between $\vec{D_1}$ and $\vec{D_2}$ to $\vec{P}$.

B.7. Intersection of a sphere and a line

For a sphere and a line to intersect, there must be a point on the line that has a distance to the center of the sphere equal to the sphere's radius.

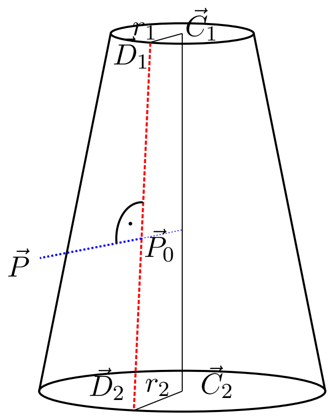


Figure B.2.: Closest point on coat of cone.

Let $\vec{c}$ denote the spheres center, r its radius and $\vec{p} + t\vec{d}$ with $t \in \mathbb{R}$ the line. Then the condition is

$$\left(\vec{p} + t\vec{d} - \vec{c}\right)^2 = r^2$$
$$\Rightarrow \qquad t = \sqrt{\frac{r^2 - (\vec{p} - \vec{c})^2 - 2\vec{d} \cdot (\vec{p} - \vec{c})}{\vec{d}^2}}$$

If the equation has results, the results have to be plugged into the line equation to find the intersection points.

B.8. Intersection of a cone and a line

This appendix is based on [33].

Without loss of generality, let the tip of the cone be at the origin. A point $\vec{x}$ is on the coat of the cone extended to infinity when

$$\vec{c} \cdot \vec{x} = \cos(\Phi)\,|\vec{p}|$$

where $\vec{c}$ is the direction of the center line and Φ is the opening angle of the cone. Let the line be described by $\vec{p} + t\vec{d}$ with $t \in \mathbb{R}$. The set of points on the line fulfilling the equation is

$$\begin{aligned}
& \left(\vec{c}\cdot\left(\vec{p}+t\vec{d}\right)\right)^2 = \cos^2(\Phi)\left(\vec{p}+t\vec{d}\right)^2 && \text{and } \left(\vec{p}+t\vec{d}\right)\cdot\vec{c}\geq 0 \\
\Leftrightarrow\quad & \left(\vec{p}+t\vec{d}\right)^T\left(\vec{c}\vec{c}^T\right)\left(\vec{p}+t\vec{d}\right) - \cos^2(\Phi)\left(\vec{p}+t\vec{d}\right)^2 = 0 && \text{and } \left(\vec{p}+t\vec{d}\right)\cdot\vec{c}\geq 0 \\
\Leftrightarrow\quad & \left(\vec{p}+t\vec{d}\right)^T M\left(\vec{p}+t\vec{d}\right) = 0 && \text{and } \left(\vec{p}+t\vec{d}\right)\cdot\vec{c}\geq 0
\end{aligned}$$

where

$$M := \vec{c}\vec{c}^T - \cos^2(\Phi)$$

Expanding the equation gives a quadratic equation in t:

$$t^2c_1 + tc_2 + c_3 = 0 \qquad\qquad \text{and } \left(\vec{p}+t\vec{d}\right)\cdot\vec{c}\geq 0$$

where

$$\begin{aligned}
c_1 &:= \vec{d}^T M\vec{d} \\
c_2 &:= 2\vec{p}^T M\vec{d} \\
c_3 &;= \vec{p}^T M\vec{p}
\end{aligned}$$

The results for the equation have to be found and determined if they fulfill $\left(\vec{p}+t\vec{d}\right)\cdot\vec{c}\geq 0$.

Bibliography

[1] A. B. Accary and E. Galin. Fast distance computation between a point and cylinders, cones, line swept spheres and cone-spheres. Technical Report RR-LIRIS-2004-020, LIRIS UMR 5205 CNRS/INSA de Lyon/Université Claude Bernard Lyon 1/Université Lumière Lyon 2/École Centrale de Lyon, May 2004. URL `http://liris.cnrs.fr/publis/?id=1929`. Validé par : comite Journal of Graphics Tools.

[2] J. Allard, S. Cotin, F. Faure, P.-J. Bensoussan, F. Poyer, C. Duriez, H. Delingette, and L. Grisoni. SOFA - an open source framework for medical simulation. In *Medicine Meets Virtual Reality (MMVR'15)*, Long Beach, USA, February 2007.

[3] S. Allner. *Echtzeitsimulation von Fäden für Nähvorgänge zum Einsatz in einem medizinischen Trainingssimulator*. Bachelorarbeit, Universität Heidelberg, Heidelberg, 2011.

[4] M. Anitescu and F. A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. *Nonlinear Dynamics*, 14:231–247, 1997. ISSN 0924-090X. doi: 10.1023/A:1008292328909.

[5] K.-i. Anjyo, Y. Usami, and T. Kurihara. A simple method for extracting the natural beauty of hair. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, page 111–120, New York, NY, USA, 1992. ACM. ISBN 0-89791-479-1. doi: 10.1145/133994.134021. URL `http://doi.acm.org/10.1145/133994.134021`.

[6] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, SIGGRAPH 1989, page 223–232, New York, NY, USA, 1989. ACM. ISBN 0-89791-312-4. doi: 10.1145/74333.74356. URL `http://doi.acm.org/10.1145/74333.74356`.

[7] D. Baraff. Curved surfaces and coherence for non-penetrating rigid body simulation. In *Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '90, page 19–28, New York, NY, USA, 1990. ACM. ISBN 0-89791-344-2. doi: 10.1145/97879.97881. URL `http://doi.acm.org/10.1145/97879.97881`.

[8] D. Baraff. Coping with friction for non-penetrating rigid body simulation. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, page 31–41, New York, NY, USA, 1991. ACM. ISBN 0-89791-436-8. doi: 10.1145/122718.122722. URL `http://doi.acm.org/10.1145/122718.122722`.

[9] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, page 23–34, New York, NY, USA, 1994. ACM. ISBN 0-89791-667-0. doi: 10.1145/192161.192168. URL `http://doi.acm.org/10.1145/192161.192168`.

[10] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, page 43–54, New York, NY, USA, 1998. ACM. ISBN 0-89791-999-8. doi: http://doi.acm.org/10.1145/280814.280821. URL `http://doi.acm.org/10.1145/280814.280821`.

[11] T. Becker. *3D-Echtzeitsimulation zum Training mikrochirurgischer Nähvorgänge*. Diploma thesis, Hochschule Mannheim, Mannheim, 2008.

[12] R. Bergamaschi. Farewell to see one, do one, teach one? *Surgical Endoscopy*, 15(7):637–637, 2001.

[13] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. Discrete elastic rods. In *ACM SIGGRAPH 2008 papers*, SIGGRAPH '08, page 63:1–63:12, New York, NY, USA, 2008. ACM. ISBN 978-1-4503-0112-1. doi: 10.1145/1399504.1360662. URL `http://doi.acm.org/10.1145/1399504.1360662`.

[14] F. Bertails, B. Audoly, M.-P. Cani, B. Querleux, F. Leroy, and J.-L. Lévêque. Super-helices for predicting the dynamics of natural hair. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, page 1180–1187, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. doi: 10.1145/1179352.1142012. URL `http://doi.acm.org/10.1145/1179352.1142012`.

[15] M. Bro-nielsen and S. Cotin. Real-time volumetric deformable models for surgery simulation using finite elements and condensation. In *Computer Graphics Forum*, page 57–66, 1996.

[16] J. Brown, K. Montgomery, J.-C. Latombe, and M. Stephanides. A microsurgery simulation system. In W. Niessen and M. Viergever, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2001*, volume 2208 of *Lecture Notes in Computer Science*, pages 137–144.

Springer Berlin Heidelberg, 2001. ISBN 978-3-540-42697-4. URL `http://dx.doi.org/10.1007/3-540-45468-3_17`.

[17] J. Brown, S. Sorkin, J.-C. Latombe, K. Montgomery, and M. Stephanides. Algorithmic tools for real-time microsurgery simulation. *Medical Image Analysis*, 6(3):289–300, September 2002. ISSN 1361-8415. doi: 10.1016/S1361-8415(02)00086-5. URL `http://www.sciencedirect.com/science/article/pii/S1361841502000865`.

[18] J. Brown, J.-C. Latombe, and K. Montgomery. Real-time knot-tying simulation. *The Visual Computer*, 20(2-3):165–179, 2004. ISSN 0178-2789. doi: 10.1007/s00371-003-0226-y. URL `http://dx.doi.org/10.1007/s00371-003-0226-y`.

[19] J. C. Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods.* Wiley-Interscience, New York, NY, USA, 1987. ISBN 0-471-91046-5.

[20] E. Catto. Iterative dynamics with temporal coherence. *Game Developer Conference*, 2005.

[21] J. T. Chang, J. Jin, and Y. Yu. A practical model for hair mutual interactions. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '02, page 73–80, New York, NY, USA, 2002. ACM. ISBN 1-58113-573-4. doi: 10.1145/545261.545273. URL `http://doi.acm.org/10.1145/545261.545273`.

[22] H. H. Chen and T. S. Huang. A survey of construction and manipulation of octrees. *Computer Vision, Graphics, and Image Processing*, 43 (3):409 – 431, 1988. ISSN 0734-189X. doi: http://dx.doi.org/10.1016/0734-189X(88)90092-8. URL `http://www.sciencedirect.com/science/article/pii/0734189X88900928`.

[23] B. Choe, M. G. Choi, and H.-S. Ko. Simulating complex hair with robust collision handling. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '05, page 153–160, New York, NY, USA, 2005. ACM. ISBN 1-59593-198-8. doi: 10.1145/1073368.1073389. URL `http://doi.acm.org/10.1145/1073368.1073389`.

[24] J. Chung and G. M. Hulbert. A time integration algorithm for structural dynamics with improved numerical dissipation: The generalized-α method. *Journal of Applied Mechanics*, 60(2):371–375, June 1993. ISSN 0021-8936. doi: 10.1115/1.2900803. URL `http://dx.doi.org/10.1115/1.2900803`.

[25] S. Cover, N. Ezquerra, J. O'Brien, R. Rowe, T. Gadacz, and E. Palm. Interactively deformable models for surgery simulation. *IEEE Computer Graphics and Applications*, 13(6):68–75, 1993. ISSN 0272-1716. doi: 10.1109/38.252559.

[26] A. Cromer. Stable solutions using the euler approximation. *American Journal of Physics*, 49(5):455–459, 1981. doi: 10.1119/1.12478. URL `http://link.aip.org/link/?AJP/49/455/1`.

[27] A. Daldegan, N. M. Thalmann, T. Kurihara, and D. Thalmann. An integrated system for modeling, animating and rendering hair. *Computer Graphics Forum*, 12(3):211–221, 1993. ISSN 1467-8659. doi: 10.1111/1467-8659.1230211. URL `http://onlinelibrary.wiley.com/doi/10.1111/1467-8659.1230211/abstract`.

[28] H. Delingette. Toward realistic soft-tissue modeling in medical simulation. *Proceedings of the IEEE*, 86(3):512–523, 1998. ISSN 0018-9219. doi: 10.1109/5.662876.

[29] J. Dequidt, D. Marchal, and L. Grisoni. Time-critical animation of deformable solids: Collision detection and deformable objects. *Comput. Animat. Virtual Worlds*, 16(3-4):177–187, July 2005. ISSN 1546-4261. doi: 10.1002/cav.v16:3/4. URL `http://dx.doi.org/10.1002/cav.v16:3/4`.

[30] M. Desbrun, P. Schröder, and A. Barr. Interactive animation of structured deformable objects. In *Proceedings of the 1999 conference on Graphics interface '99*, page 1–8, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-632-7. URL `http://portal.acm.org/citation.cfm?id=351631.351638`.

[31] C. Duriez, C. Andriot, and A. Kheddar. Signorini's contact model for deformable objects in haptic simulations. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings*, volume 4, pages 3232–3237 vol.4, 2004. doi: 10.1109/IROS.2004.1389915.

[32] B. C. Eaves. The linear complementarity problem. *Management Science*, 17 (9):612–634, 1971. doi: 10.1287/mnsc.17.9.612.

[33] D. Eberly. Intersection of a line and a cone, October 2000. URL `http://www.geometrictools.com/`.

[34] D. Eberly. Distance between two line segments in 3D, November 2008. URL `http://www.geometrictools.com/Documentation/DistanceLine3Line3.pdf`.

[35] D. Eberly. Distance between point and line, ray, or line segment, January 2008. URL `http://www.geometrictools.com/Documentation/DistancePointLine.pdf`.

[36] E. Erel, B. Aiyenibe, and P. E. M. Butler. Microsurgery simulators in virtual reality: review. *Microsurgery*, 23(2):147–152, 2003. ISSN 0738-1085. doi: 10.1002/micr.10106. PMID: 12740888.

[37] C. Ericson. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004. ISBN 1558607323.

[38] K. Erleben. Velocity-based shock propagation for multibody dynamics animation. *ACM Trans. Graph.*, 26(2), June 2007. ISSN 0730-0301. doi: 10.1145/1243980.1243986. URL `http://doi.acm.org/10.1145/1243980.1243986`.

[39] J. I. Fann, A. D. Caffarelli, G. Georgette, S. K. Howard, D. M. Gaba, P. Youngblood, R. S. Mitchell, and T. A. Burdon. Improvement in coronary anastomosis with cardiac surgery simulation. *The Journal of thoracic and cardiovascular surgery*, 136(6):1486–1491, December 2008. ISSN 1097-685X. doi: 10.1016/j.jtcvs.2008.08.016. PMID: 19114195.

[40] S. P. Fanua, J. Kim, and E. Shaw Wilgis. Alternative model for teaching microsurgery. *Microsurgery*, 21(8):379–382, 2001. ISSN 1098-2752. doi: 10.1002/micr.21812. URL `http://dx.doi.org/10.1002/micr.21812`.

[41] R. Featherstone. *Rigid body dynamics algorithms*. Springer, New York, 2008. ISBN 9780387743158 0387743154 0387743146 9780387743141. URL `http://public.eblib.com/EBLPublic/PublicView.do?ptiID=372684`.

[42] P. J. Figueras Sola, S. Rodriguez Bescós, P. Lamata, J. B. Pagador, F. M. Sánchez-Margallo, and E. J. Gómez. Virtual reality thread simulation for laparoscopic suturing training. *Studies in health technology and informatics*, 119:144–149, 2006. ISSN 0926-9630. PMID: 16404034.

[43] M. Gissler, M. Becker, and M. Teschner. Local constraint methods for deformable objects. In *Proceedings of the 3rd Workshop in VR Interactions and Physical Simulation (VRIPHYS)*, page 1–8, 2006.

[44] R. Goldenthal, D. Harmon, R. Fattal, M. Bercovier, and E. Grinspun. Efficient simulation of inextensible cloth. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):to appear, 2007.

[45] R. E. Goldstein and S. A. Langer. Nonlinear dynamics of stiff polymers. *Physical Review Letters*, 75(6):1094–1097, August 1995. doi: 10.1103/PhysRevLett.75.1094. URL `http://link.aps.org/doi/10.1103/PhysRevLett.75.1094`.

[46] P. J. Gorman, A. H. Meier, and T. M. Krummel. Simulation and virtual reality in surgical education: real or unreal? *Archives of Surgery*, 134(11):1203, 1999.

[47] S. Green, G. Turkiyyah, and D. Storti. Subdivision-based multilevel methods for large scale engineering simulation of thin shells. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, SMA '02, page 265–272, New York, NY, USA, 2002. ACM. ISBN 1-58113-506-8. doi: 10.1145/566282.566321. URL `http://doi.acm.org/10.1145/566282.566321`.

[48] M. Grégoire and E. Schömer. Interactive simulation of one-dimensional flexible parts. *Computer-Aided Design*, 39(8):694–707, August 2007. ISSN 0010-4485. doi: 10.1016/j.cad.2007.05.005. URL `http://www.sciencedirect.com/science/article/pii/S0010448507001169`.

[49] E. Guendelman, R. Bridson, and R. Fedkiw. Nonconvex rigid bodies with stacking. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, page 871–878, New York, NY, USA, 2003. ACM. ISBN 1-58113-709-5. doi: 10.1145/1201775.882358. URL `http://doi.acm.org/10.1145/1201775.882358`.

[50] S. Hadap. Oriented strands: dynamics of stiff multi-body system. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '06, page 91–100, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association. ISBN 3-905673-34-7. URL `http://dl.acm.org/citation.cfm?id=1218064.1218077`.

[51] S. Hadap, M.-P. Cani, M. Lin, T.-Y. Kim, F. Bertails, S. Marschner, K. Ward, and Z. Kačić-Alesić. Strands and hair: modeling, animation, and rendering. In *ACM SIGGRAPH 2007 courses*, SIGGRAPH '07, page 1–150, New York, NY, USA, 2007. ACM. ISBN 978-1-4503-1823-5. doi: 10.1145/1281500.1281689. URL `http://doi.acm.org/10.1145/1281500.1281689`.

[52] J. K. Hahn. Realistic animation of rigid bodies. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, page 299–308, New York, NY, USA, 1988. ACM. ISBN 0-89791-275-6. doi: 10.1145/54852.378530. URL `http://doi.acm.org/10.1145/54852.378530`.

[53] L. Hilde, P. Meseure, and C. Chaillou. A fast implicit integration method for solving dynamic equations of movement. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '01, page 71–76,

New York, NY, USA, 2001. ACM. ISBN 1-58113-427-4. doi: 10.1145/505008.505022. URL `http://doi.acm.org/10.1145/505008.505022`.

[54] R. Holbrey, A. Bulpitt, K. Brodlie, M. Walkley, and J. Scott. A model for virtual suturing in vascular surgery. In *Theory and Practice of Computer Graphics, 2004. Proceedings*, pages 50–58, 2004. doi: 10.1109/TPCG.2004.1314452.

[55] R. P. Holbrey. *Virtual suturing for training in vascular surgery.* Diss., University of Leeds, November 2004. URL `http://etheses.whiterose.ac.uk/1326/`.

[56] N. Hüsken, O. Schuppe, E. Sismanidis, and F. Beier. MicroSim - a microsurgical training simulator. In *Medicine Meets Virtual Reality 20*, volume 20. IOS Press, January 2013.

[57] P. Jung, S. Leyendecker, J. Linn, and M. Ortiz. Discrete lagrangian mechanics and geometrically exact cosserat rods. Technical Report 160, Fraunhofer (ITWM), 2009.

[58] D. M. Kaufman, T. Edmunds, and D. K. Pai. Fast frictional dynamics for rigid bodies. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 946–956, New York, NY, USA, 2005. ACM. doi: 10.1145/1186822.1073295. URL `http://doi.acm.org/10.1145/1186822.1073295`.

[59] R. Keiser, M. Müller, B. Heidelberger, M. Teschner, and M. H. Gross. Contact handling for deformable point-based objects. In *VMV*, pages 315–322, 2004.

[60] I. Klapper. Biological applications of the dynamics of twisted elastic rods. *Journal of Computational Physics*, 125(2):325–337, May 1996. ISSN 0021-9991. doi: 10.1006/jcph.1996.0097. URL `http://www.sciencedirect.com/science/article/pii/S0021999196900972`.

[61] B. Kubiak, N. Pietroni, F. Ganovelli, and M. Fratarcangeli. A robust method for real-time thread simulation. In *Proceedings of the 2007 ACM symposium on Virtual reality software and technology*, VRST '07, page 85–88, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-863-3. doi: 10.1145/1315184.1315198. URL `http://doi.acm.org/10.1145/1315184.1315198`.

[62] C. Lacoursiere. Splitting methods for dry frictional contact problems in rigid multibody systems: Preliminary performance results. In *Conference Proceedings from SIGRAD2003*, page 11–16, 2003.

[63] H. Lang, J. Linn, and M. Arnold. Multi-body dynamics simulation of geometrically exact cosserat rods. *Multibody System Dynamics*, 25(3):285–312, 2011.

[64] J. Langer and D. Singer. Lagrangian aspects of the kirchhoff elastic rod. *SIAM Review*, 38(4):605–618, 1996. doi: 10.1137/S0036144593253290. URL `http://epubs.siam.org/doi/abs/10.1137/S0036144593253290`.

[65] D. A. Lannon, J.-A. Atkins, and P. E. Butler. Non-vital, prosthetic, and virtual reality models of microsurgical training. *Microsurgery*, 21(8):389–393, 2001. ISSN 1098-2752. doi: 10.1002/micr.21709. URL `http://dx.doi.org/10.1002/micr.21709`.

[66] K. Larsson, G. Wallgren, and M. G. Larson. Interactive simulation of a continuum mechanics based torsional thread. In *Vriphys 10: 7th workshop on virtual reality interaction and physical simulation*, page 49–58, 2010.

[67] J. Lenoir, P. Meseure, L. Grisoni, and C. Chaillou. Surgical thread simulation. *ESAIM: Proceedings*, 12:102–107, 2002. ISSN 1270-900X. doi: 10.1051/proc:2002017. URL `http://www.esaim-proc.org/articles/proc/pdf/2002/02/lenoir.pdf`.

[68] J. Lloyd. Fast implementation of lemke's algorithm for rigid body contact simulation. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005*, pages 4538–4543, 2005. doi: 10.1109/ROBOT.2005.1570819.

[69] P. Lötstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal on Scientific and Statistical Computing*, 5(2):370–393, June 1984. ISSN 0196-5204, 2168-3417. doi: 10.1137/0905028. URL `http://epubs.siam.org/doi/abs/10.1137/0905028?journalCode=sijcd4`.

[70] S. Martin, P. Kaufmann, M. Botsch, E. Grinspun, and M. Gross. Unified simulation of elastic rods, shells, and solids. In *ACM SIGGRAPH 2010 papers*, SIGGRAPH '10, page 39:1–39:10, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0210-4. doi: 10.1145/1833349.1778776. URL `http://doi.acm.org/10.1145/1833349.1778776`.

[71] W. Mason and P. Strike. Short communication see one, do one, teach one-is this still how it works? a comparison of the medical and nursing professions in the teaching of practical procedures. *Medical teacher*, 25(6):664–666, 2003.

[72] A. Miehlke. *Illustrierte Geschichte der Mikrochirurgie: die historische Entwicklung in den verschiedenen operativen Disziplinen.* Voltmedia, Paderborn, 2007. ISBN 3867632006 9783867632003.

[73] V. J. Milenkovic. Position-based physics: Simulating the motion of many highly interacting spheres and polyhedra. In *In Computer Graphics*, page 129–136. ACM Press, 1996.

[74] V. J. Milenkovic and H. Schmidl. Optimization-based animation. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, page 37–46, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383263. URL `http://doi.acm.org/10.1145/383259.383263`.

[75] J. Mosegaard, P. Herborg, and T. S. Sørensen. A GPU accelerated spring mass system for surgical simulation. *Studies in health technology and informatics*, 111:342–348, 2005. ISSN 0926-9630. PMID: 15718756.

[76] M. Mueller and M. Teschner. Volumetric meshes for real-time medical simulations. In *Proceedings BVM '03*, page 279–283, 2003.

[77] M. Müller, B. Heidelberger, M. Hennix, and J. Ratcliff. Position based dynamics. *J. Vis. Comun. Image Represent.*, 18(2):109–118, April 2007. ISSN 1047-3203. doi: 10.1016/j.jvcir.2007.01.005. URL `http://dx.doi.org/10.1016/j.jvcir.2007.01.005`.

[78] M. Müller, T.-Y. Kim, and N. Chentanez. Fast simulation of inextensible hair and fur. In *VRIPHYS'12*, page 39–44, 2012.

[79] A. Nealen, M. Müller, R. Keiser, E. Boxerman, and M. Carlson. Physically based deformable models in computer graphics. *Computer Graphics Forum*, 25(4):809–836, 2006. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2006.01000.x. URL `http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2006.01000.x/abstract`.

[80] J. Niiranen. Fast and accurate symmetric euler algorithm for electromechanical simulations. In *IMACS*, pages I–71, 1999.

[81] T. P. Olson, Y. T. Becker, R. McDonald, and J. Gould. A simulation-based curriculum can be used to teach open intestinal anastomosis. *The Journal of surgical research*, 172(1):53–58, January 2012. ISSN 1095-8673. doi: 10.1016/j.jss.2010.08.009. PMID: 20864120.

[82] R. V. O'Toole, R. R. Playter, T. M. Krummel, W. C. Blank, N. H. Cornelius, W. R. Roberts, W. J. Bell, and M. Raibert. Measuring and developing suturing technique with a virtual reality surgical simulator. *Journal of the American College of Surgeons*, 189(1):114–127, July 1999. ISSN 1072-7515. PMID: 10401747.

[83] D. K. Pai. Strands: Interactive simulation of thin solids using cosserat models. In *Computer Graphics Forum*, volume 21, page 347–352. Blackwell Publishing, Inc, 2002.

[84] M. Pauly, D. K. Pai, and L. J. Guibas. Quasi-rigid objects in contact. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, SCA '04, page 109–119, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association. ISBN 3-905673-14-2. doi: 10.1145/1028523.1028539. URL `http://dx.doi.org/10.1145/1028523.1028539`.

[85] J. Phillips, A. Ladd, and L. E. Kavraki. Simulated knot tying. In *In Proceedings of the IEEE International Conference on Robotics and Automation*, page 841–846, 2002.

[86] E. Plante, M.-P. Cani, and P. Poulin. A layered wisp model for simulating interactions inside long hair. In N. Magnenat-Thalmann and D. Thalmann, editors, *Computer Animation and Simulation 2001*, Eurographics, pages 139–148. Springer Vienna, 2001. ISBN 978-3-211-83711-5. URL `http://dx.doi.org/10.1007/978-3-7091-6240-8_13`.

[87] M. Poulsen, S. M. Niebe, and K. Erleben. Heuristic convergence rate improvements of the projected gauss-seidel method for frictional contact problems. In *WSCG 2010*, pages 135–142, Plzen, 2010. ISBN 9788086943886.

[88] W. H. Press. *Numerical recipes: the art of scientific computing.* Cambridge University Press, Cambridge, UK; New York, 2007. ISBN 0521880688.

[89] S. Punak and S. Kurenov. Simplified cosserat rod for interactive suture modeling. In *Medicine Meets Virtual Reality 18*. IOS Press, February 2011.

[90] H. Qin and D. Terzopoulos. D-NURBS: a physics-based framework for geometric design. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):85–96, 1996. ISSN 1077-2626. doi: 10.1109/2945.489389.

[91] Y. Rémion, J.-M. Nourrit, and D. Gillard. A dynamic animation engine for generic spline objects. *The Journal of Visualization and Computer Animation*, 11(1):17–26, 2000. ISSN 1099-1778. doi: 10.1002/(SICI)1099-1778(200002)11:1⟨17::AID-VIS213⟩3.0.CO;2-9. URL `http://dx.doi.org/10.1002/(SICI)1099-1778(200002)11:1<17::AID-VIS213>3.0.CO;2-9`.

[92] R. E. Rosenblum, W. E. Carlson, and E. 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. ISSN 1099-1778. doi: 10.1002/vis.4340020410. URL `http://onlinelibrary.wiley.com/doi/10.1002/vis.4340020410/abstract`.

[93] W. Rungjiratananon, Y. Kanamori, and T. Nishita. Elastic rod simulation by chain shape matching with twisting effect. In *ACM SIGGRAPH ASIA 2010 Sketches*, SA '10, page 27:1–27:2, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0523-5. doi: 10.1145/1899950.1899977. URL `http://doi.acm.org/10.1145/1899950.1899977`.

[94] G. Salvendy and J. Pilitsis. The development and validation of an analytical training program for medical suturing. *Human factors*, 22(2):153–170, April 1980. ISSN 0018-7208. PMID: 6993341.

[95] H. Schmidl. *Optimization-based animation.* Diss., University of Miami, 2002.

[96] O. Schuppe. An optical tracking system for a microsurgical training simulator. *Studies in health technology and informatics*, 173:445–449, 2012.

[97] A. Selle, M. Lentine, and R. Fedkiw. A mass spring model for hair simulation. *ACM Trans. Graph.*, 27(3):64:1–64:11, August 2008. ISSN 0730-0301. doi: 10.1145/1360612.1360663. URL `http://doi.acm.org/10.1145/1360612.1360663`.

[98] M. Servin and C. Lacoursière. Rigid body cable for virtual environments. *IEEE transactions on visualization and computer graphics*, 14(4):783–796, August 2008. ISSN 1077-2626. doi: 10.1109/TVCG.2007.70629. PMID: 18467754.

[99] V. Y. Sigounas, P. W. Callas, C. Nicholas, J. E. Adams, D. J. Bertges, A. C. Stanley, G. Steinthorsson, and M. A. Ricci. Evaluation of simulation-based training model on vascular anastomotic skills for surgical residents. *Simulation in Healthcare: The Journal of the Society for Simulation in Healthcare*, 7(6):334–338, December 2012. ISSN 1559-2332. doi: 10.1097/SIH.0b013e318264655e. URL `http://journals.lww.com/simulationinhealthcare/Abstract/2012/12000/Evaluation_of_Simulation_Based_Training_Model_on.2.aspx`.

[100] M. Silcowitz-Hansen, S. Niebe, and K. Erleben. Nonsmooth newton method for fischer function reformulation of contact force problems for interactive rigid body simulation. In *VRIPHYS'09*, pages 105–114, 2009.

[101] M. Silcowitz-Hansen, S. Niebe, and K. Erleben. A nonsmooth nonlinear conjugate gradient method for interactive contact force problems. *The Visual Computer*, 26(6):893–901, 2010. ISSN 0178-2789. URL `http://dx.doi.org/10.1007/s00371-010-0502-6`. 10.1007/s00371-010-0502-6.

[102] E. Sismanidis. Echtzeitinteratktion und -simulation deformierbarer dreidimensionaler objekte für mikrochirurgische trainingsmodule. Diss., 2013.

[103] G. Sobottka, T. Lay, and A. W. 0004. Stable integration of the dynamic cosserat equations with application to hair modeling. *Journal of WSCG*, 16 (1-3):73–80, 2008. URL `http://dblp.uni-trier.de/db/journals/jwscg/jwscg16.html#SobottkaL008`.

[104] J. Spillmann and M. 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*, SCA '07, page 63–72, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association. ISBN 978-1-59593-624-0. URL `http://dl.acm.org/citation.cfm?id=1272690.1272700`.

[105] J. Spillmann and M. Teschner. An adaptive contact model for the robust simulation of knots. *Computer Graphics Forum*, 27(2):497–506, 2008. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2008.01147.x. URL `http://onlinelibrary.wiley.com/doi/10.1111/j.1467-8659.2008.01147.x/abstract`.

[106] D. Stewart and J. Trinkle. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 162–169 vol.1, 2000. doi: 10.1109/ROBOT.2000.844054.

[107] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson. A computer simulation method for the calculation of equilibrium constants for the formation of physical clusters of molecules: Application to small water clusters. *The Journal of Chemical Physics*, 76(1):637–649, 1982. doi: 10.1063/1.442716. URL `http://link.aip.org/link/?JCP/76/637/1`.

[108] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, November 1988. ISSN 0178-2789, 1432-2315. doi: 10.1007/BF01908877. URL `http://link.springer.com/article/10.1007/BF01908877`.

[109] D. Terzopoulos and K. Fleischer. Modeling inelastic deformation: viscolelasticity, plasticity, fracture. In *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '88, page 269–278, New

York, NY, USA, 1988. ACM. ISBN 0-89791-275-6. doi: 10.1145/54852.378522. URL http://doi.acm.org/10.1145/54852.378522.

[110] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '87, page 205–214, New York, NY, USA, 1987. ACM. ISBN 0-89791-227-6. doi: 10.1145/37401.37427. URL http://doi.acm.org/10.1145/37401.37427.

[111] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross. Optimized spatial hashing for collision detection of deformable objects. In *VMV'03*, pages 47–54, 2003.

[112] A. Theetten, L. Grisoni, C. Andriot, and B. Barsky. Geometrically exact dynamic splines. *Computer Aided Design*, 40(1):35–48, January 2008. ISSN 0010-4485. doi: 10.1016/j.cad.2007.05.008. URL http://dx.doi.org/10.1016/j.cad.2007.05.008.

[113] A. Theetten, L. Grisoni, C. Duriez, and X. Merlhiot. Quasi-dynamic splines. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, SPM '07, page 409–414, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-666-0. doi: 10.1145/1236246.1236305. URL http://doi.acm.org/10.1145/1236246.1236305.

[114] J. Thorson. *Gaussian Elimination on a Banded Matrix.* Stanford Exploration Project, 1979. Published: Stanford Exploration Project.

[115] L. Verlet. Computer "Experiments" on classical fluids. i. Themodynamical properties of lennard-jones molecules. *Physical Review*, 159(1):98–103, 1967.

[116] H. Wakamatsu and S. Hirai. Static modeling of linear object deformation based on differential geometry. *The International Journal of Robotics Research*, 23(3):293–311, March 2004. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364904041882. URL http://ijr.sagepub.com/content/23/3/293.

[117] F. Wang, E. Burdet, A. Dhanik, T. Poston, and C. 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, 2005. doi: 10.1109/WHC.2005.44.

[118] F. Wang, E. Su, E. Burdet, and H. Bleuler. Development of a microsurgery training system. *Proceedings of the annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in*

Medicine and Biology Society, 2008:1935–1938, 2008. ISSN 1557-170X. doi: 10.1109/IEMBS.2008.4649566. PMID: 19163069.

[119] K. Ward, F. Bertails, T.-Y. Kim, S. Marschner, M.-P. Cani, and M. Lin. A survey on hair modeling: Styling, simulation, and rendering. *IEEE Transactions on Visualization and Computer Graphics*, 13(2):213–234, 2007. ISSN 1077-2626. doi: 10.1109/TVCG.2007.30.

[120] R. W. Webster, D. I. Zimmerman, B. J. Mohler, M. G. Melkonian, and R. S. Haluck. A prototype haptic suturing simulator. *Studies in health technology and informatics*, 81:567–569, 2001. ISSN 0926-9630. PMID: 11317811.

[121] R. Weinstein, J. Teran, and R. Fedkiw. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE TVCG*, page 365–374, 2006.

[122] Y. Yang, I. Tobias, and W. K. Olson. Finite element analysis of DNA supercoiling. *The Journal of Chemical Physics*, 98(2):1673–1686, January 1993. ISSN 00219606. doi: doi:10.1063/1.464283.

[123] A. Ziv, S. D. Small, and P. R. Wolpe. Patient safety and simulation-based medical education. *Medical teacher*, 22(5):489–495, 2000.