COMPUTATION OF FILAMENT WINDING PATHS WITH CONCAVITIES AND FRICTION

A Dissertation

by

HANG LI

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

DOCTOR OF PHILOSOPHY

Chair of Committee,	John Keyser
Committee Members,	Scott Schaefer
	Shinjiro Sueda
	Ergun Akleman
Head of Department,	Scott Schaefer

May 2021

Major Subject: Computer Science

ABSTRACT

We introduce an efficient method to support generation of geometric winding paths on parametric shapes. Filament winding is a technology for producing composite materials by winding resin-infused fibers around the underlying model. While filament winding is a long-standing manufacturing method, only a few shapes, primarily cylinders, have been manufactured in practice. Extending this to a broader range of parametric surfaces is desirable.

For convex objects without friction, generating a winding path over a model is equivalent to finding a locally geodesic path on the surface. We propose a physically-based method ideally suited for generating these geodesics, and show how it can be augmented to incorporate friction in the simulation process. For non-convex objects, it is important to correctly handle the bridging of filaments across local concavities. We therefore propose an efficient method for lifting a filament from and returning it to a surface, within the same simulation framework.

We demonstrate how this method forms the basis for an end-to-end system that designers can use to create, visualize, and redesign winding paths for a variety of shapes.

DEDICATION

To my mother, my father, and my wife.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Dr. John Keyser, for his invaluable guidance, patience, and expertise throughout my doctoral studies. Without his consistent support and encouragement, I would not have made it.

I would like to thank my advisory committee members, Dr. Scott Schaefer, Dr. Shinjiro Sueda, and Dr. Ergun Akleman. They provided me with extensive professional advice and insightful feedback about scientific research.

I am grateful for the opportunity to study in the Department of Computer Science and Engineering. It was a memorable experience to work with everyone here.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a dissertation committee consisting of Dr. John Keyser, Dr. Scott Schaefer, Dr. Shinjiro Sueda, and Dr. Ergun Akleman.

All work conducted for the dissertation was completed by the student under Dr. John Keyser and Dr. Shinjiro Sueda's supervision.

Funding Sources

Graduate study was supported by the Department of Computer Science and Engineering at Texas A&M University.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

This work deals with the generation of paths formed by filaments as they are pulled over a surface in tension and under friction. This work is motivated by the manufacturing process of filament winding. Our path generation approach is an important step toward expanding the range of shapes that can be created with this process.

1.1 Motivating Problem

Filament winding is a long-standing method used for manufacturing composite materials, particularly tanks, pipes, and other containers [24]. The basic process involves winding filaments with embedded resins around a shape, then curing that shape to produce a strong (yet usually thin and light) object. The base shape is referred to as a mandrel, and it is usually mounted on a machine that spins the mandrel about a single axis. While the mandrel is rotated continuously, a movable head dispenses filaments under tension, and the head is moved (typically either with 2 or 5 degrees of freedom) so that the fibers are gradually wound around the entire object. Fig. 1.1 illustrates how a winding machine works. There are usually two families of fibers (cyan and green) which are wound in different directions. In Fig. 1.1 the blue fiber is the one being wound. Q represents the head position and P the contact point on the mandrel. l is the axis the mandrel is rotating around.

Figure 1.1: An example of filament winding on a cylinder.

However, to this point, most of the practical work in winding has been to manufacture only a very limited range of objects, with cylinders (often for gas and liquid storage tanks) by far the

most common shape. Some previous works have developed approaches allowing for winding around T-junctions of cylinders [33], winding around a torus [22, 45], winding around elbows (bends in a cylinder) often based on the torus formulation [18, 19, 43], or winding over a "dome" shape that might cap a cylinder [44]. The range of shapes has remained very limited. Our goal is to significantly expand the range of shapes that it is feasible to use filament winding for, by generating winding paths for more general shapes and by providing analysis tools that identify when a shape is feasible to wind, or how it might be modified to become more feasible for winding.

1.2 Filament Winding Paths and Geodesics

For strength, fibers are typically wound around the object in multiple directions, so that different layers of filaments cross over each other. In order to manufacture an object, appropriate winding paths must be determined that allow the filaments to cover the surface of the mandrel. For cylindrical shapes, the questions to be answered are usually just the angle to wind the filaments over the surface, considering the width of filaments laid down at a time, the desired angle at which fibers cross, and the number of layers needed. For a more general shape, there are more significant challenges in determining feasible winding paths, beginning with determining whether winding is even possible.

The paths that a filament can take when being wound around a surface are strongly connected to the geometry of the surface. Since the filaments are laid down on the surface in tension, if the surface is completely smooth then the filaments must follow a geodesic path on the surface. That is, when determining a completely feasible winding path, one must identify a geodesic path on the surface. Incorporating friction opens the possibility of non-geodesic winding paths.

Feasibility requires more than just geodesic (or near-geodesic) properties, however. In particular, the filament must travel over the surface in a direction where the surface is locally convex in the direction of the filament. Because the filaments are under tension, they will "bridge" across any concave region of the surface. Fig. 1.2 illustrates this, where the green path is feasible since the geodesic is convex along the curve. The red path, though geodesic, is not a feasible solution because of its local concavity. In manufacturing, if the winding machine follows the red path, the

winding result will end up with the blue curve, which is partially off the surface.

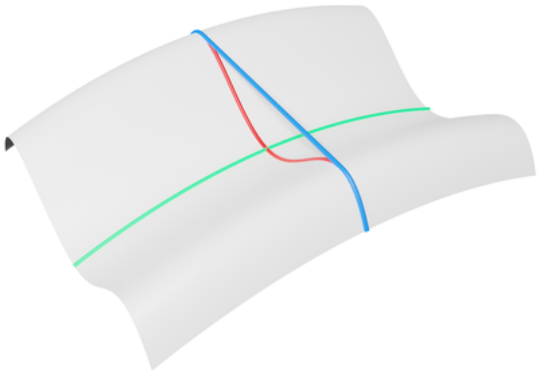
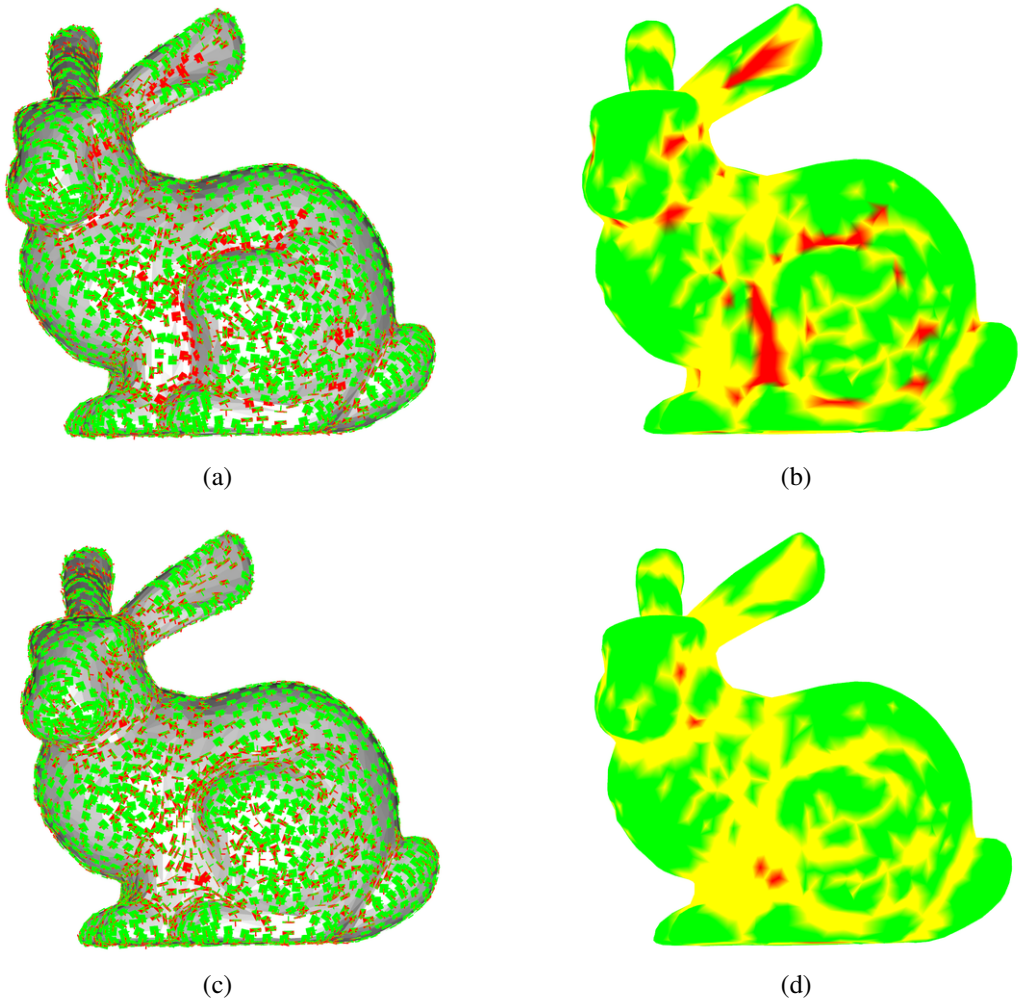


Figure 1.2: An example of convex and concave geodesic paths.

Note, then, that convex regions of the surface can be locally wound in any directions, fully concave regions cannot be wound, and saddle regions can be wound in some directions but not others. Fig. 1.3 shows an example. In (a) and (c), each rectangle represents a vertex. Green parts on a rectangle represent convex directions, which are possible for winding. Red parts represent concave directions, which are impossible for winding. In (b) and (d), green means convex points, yellow means saddle points, and red means concave points. (a) and (b) use an identical model while (c) and (d) use a different one. Obviously, if a vertex in (a) has an all-green or all-red rectangle, it is in the green or red area in (b). If the vertex's corresponding rectangle contains both red and green, it is in the yellow area. The same is true for (c) and (d).

A concave point is impossible to wind regardless of the direction of the winding path. But for a saddle point, the ability to wind depends on the curve's direction. We can modify the shape and minimize the concave area by popping out concave vertices. In Fig. 1.3, the model in (c) and (d) are modified from (a) and (b).

In a practical setting, there are other considerations for valid winding paths, such as ensuring that the head can maneuver appropriately without colliding with the mandrel. These considerations are global, scale-dependent, and specific to the physical winding machine, rather than a fundamen-



(a) (b)

(c) (d)

Figure 1.3: Possible winding directions of a model. (a) and (c) show the possible directions (convex, green) and impossible directions (concave, red) for filament winding. (b) and (d) show the convex points (green), concave points (red), and saddle points (yellow) on the surface. The model in (c) and (d) is a modified version of the one in (a) and (b), where some concave vertices have been pushed outward.

tal characteristic of the surface shape itself. So, we will ignore these considerations for the purpose of this work.

For a variety of reasons, people have also been exploring non-geodesic paths for filament winding. This is particularly motivated by desire to adjust the angles of fiber crossings, improve coverage of the mandrel shape, and obtain correspondence with principal lines and designed path layouts. Non-geodesic paths rely on the ability of the surface to resist slipping as the filaments are wound over them, and the extent to which this is really feasible is dependent on materials and the setup of the machine itself (tension plays a role). Geodesic paths are therefore ideal, and the closer a non-geodesic path is to geodesic, the better.

Producing a "good" filament-wound part involves many other considerations, such as generating pairs of winding paths that cross at larger angles (for greater strength) and ensuring density of fiber coverage of the mandrel surface. In this work, we demonstrate how our method can be used in an end-to-end system that can provide a designer feedback in such ways.

Our goal is to generate winding paths for a wide class of shapes, such that a designer can understand what winding is produced for a given shape, and can modify the shape design as a result. As a result, our work in this work is focused on finding paths that closely follow geodesic curves on the surface, and to do so while considering convexity of the surface along the winding path. We wish to do this for a wide range of surfaces and shapes.

1.3 Thesis

The filament winding problem can be solved with physically based fiber simulation in the 2D parametric space. With the method we propose, the winding solution can be both accurate and fast to generate.

To achieve this goal, we will introduce a physically-based model for geodesic solving on parametric surfaces. Our model will be simulated in the 2D parametric space. In real-world manufacturing, a winding path is influenced by friction and may not be an exact geodesic. We plan to include friction in our method. Due to local concavities, winding path bridging may happen in the winding process. We will develop an algorithm to handle bridging.

Winding path evaluation can be used to provide feedback to filament winding designers. So we will propose several analysis methods to help improve winding paths. To demonstrate the possibility of applying our method in real-world manufacturing, we present an end-to-end system for filament winding design. Besides parametric surfaces, there are many other geometry representation methods that are widely used. So we will also apply our method to a wider range of geometries such as free-form meshes.

1.4 Contributions and Organization

We present a new approach for finding feasible winding paths across arbitrary parametric surfaces. Specifically, we present:

- a physics-based model of the filament motion under tension that can be solved for dynamically in 2D and incorporation of friction in the filament behavior model (Chapter 4);
- a method for dealing with bridging behavior across concave areas within the same framework (Chapter 5);
- methods for winding path analysis and evaluation, an end-to-end filament winding design system, and an automatic shape editing method (Chapter 6);
- filament winding examples generated by our method including freeform meshes (Chapter 7).

2. BACKGROUND WORK

The generation of paths on surfaces has a number of applications, and this has motivated a great deal of study within the graphics community. We discuss three areas of related work: work in filament winding, work in computing geodesic paths on surfaces, and work in physically-based modeling.

2.1 Filament Winding

Filament winding has a long history, with the technique going back over 50 years [31]. The basic process is described in a number of books related to manufacture of composite materials [24]. Research in filament winding has ranged from computation of winding paths and shape generation (the focus of our work) to work on materials, machine construction, and machine operation to realize a given path. Research in the topic has been published in a wide range of forums in fields including materials science and various engineering disciplines, and it is still a topic of active interest [36].

In practice, the use of filament winding is overwhelmingly oriented toward cylindrical objects or closely related shapes such as domes (to cap cylinders) [20, 44], tori (to connect cylinders) and elbows (to change cylinder directions) [18, 19, 43, 22, 45], and cylinder junctions [33, 32]. Filament winding has also been extended to some surfaces of revolution [1, 17, 39, 13]. Costalonga Martins et al. [7] introduce a core-less filament winding method, which requires post-winding processing.

We are not aware of any prior work that deals with general parametric shapes or arbitrary winding paths as we do. The closest related prior work is that of Fu et al. [12]. Like our work, their method can deal with a wider range of shapes than other filament winding work, includes consideration of friction, and can detect bridging in a path. However, the filament winding is not analyzed as a process (just a path that is geometrically updated), the friction is used only for final analysis and not during the simulation, and the bridging is detected only as a collision/condition to

be avoided, so is neither representable in the final path nor able to be maintained in intermediate steps to obtain different final paths. Most importantly, their shapes are limited to nearly axisymmetric models. There is prior work on optimization of filament paths that, though different from our approach, nevertheless has a similar goal of improving a winding path from a given path [9].

2.2 Geodesic and Curve Computation on Surfaces

Since filament winding paths without friction are geodesics, much of the goal of winding path generation becomes finding geodesic or near-geodesic paths on surfaces. Most work in this area has come from the geometric modeling and computer graphics communities.

Computing geodesics on surfaces, polyhedra, and meshes [6, 16, 15, 23] has a long history. For a mesh, the computation of geodesic curvature and other discrete geometric values opened the door to a wide range of mesh-based geometric calculation [26].

A great deal of work has been done in computing geodesic paths on surfaces, usually focused on finding shortest paths across a manifold from some starting point [38, 41, 8, 42, 27]. Methods for doing this have ranged from computational geometry calculations to solutions to a heat equation. Although our goal is quite different, some of the techniques for computing geodesics are related to our own methods, though a notable difference is that our approach requires us to consider surface convexity (and, related, bridging), which is not an issue in most other applications where curves are constrained to the surface.

More generally, all geodesic calculations can be seen as a form of energy-minimizing curve on a manifold [14]. Our basic curve model (without bridging) fits into this framework, though the frictional consideration would generally be more difficult to incorporate, since it counteracts minimization.

Families of geodesic-based curves can be used to "ribbonize" a shape [30, 25], or compute weaving "foliations" [40]. They have also been used to compute manufacturing and physical behavior of models [28, 29]. Though the goals of these prior studies are quite different from ours, families of geodesic (or approximately geodesic) curves are also produced by filament winding.

Our own approach tries to improve paths to make them more geodesic, which has also been a

topic of prior research [21].

2.3 Physically-Based Filament Simulation

There has been significant previous work in physics-based simulation, and we borrow many of these ideas in our framework. Our method's focus is on generating a filament path, and not the animation of the filament winding process. But, we obtain the path through a simulation of a strand moving on a surface, and thus filament winding could be viewed similar to simulating hair [34], discrete elastic rods [3], rope [5], etc. In particular, it can be viewed as a type of highly-constrained strand in contact with a surface and in tension [37].

Our work is most notably different from the prior work in two important ways. First, our problem is defined solely in the context of a *parametric surface*. This enables us to make several computational simplifications in the physics model, in particular in projecting the forces into the tangent plane. For free-form meshes, there are plenty of previous works about mesh parameterization [11, 35]. With any of those method, a free-form mesh can be converted into a parametric surface and then apply our method to generate winding paths. Second, we can safely *ignore the twisting behavior* of the filament, greatly simplifying the formulation. These two differences allow use to derive a concise and efficient simulation model for the winding path process.

3. GEOMETRY AND PHYSICS IN FILAMENT WINDING

Parametric surfaces, including NURBS, are widely used in shape design and modeling. So, we focus on solving the geodesic and filament winding problem on parametric surfaces in this work. We describe in this chapter some fundamental definitions and characteristics of winding paths; although this formulation is our own, the concepts have been used in prior work.

3.1 Physical Constraints for Filament Winding

Suppose the mandrel shape is defined as a parametric surface $\vec{s}(u, v)$, and the filament winding path is a parametric curve $\vec{r}(t) = \vec{s}(u(t), v(t))$ defined on $\vec{s}(u, v)$. Without loss of generality, we assume that $\vec{r}(t)$ is the arc-length parameterization of the curve.

Fig. 3.1 shows the force analysis on a small piece of fiber from t_0 to $t_1 = t_0 + \Delta t$ (the blue curve). In a stable situation, all the forces are in balance. $\vec{N}$ is the support force; $\vec{T_0}$ and $\vec{T_1}$ are the tension forces; $\vec{fr}$ is the friction. The surface's normal is $\vec{n}_s(u, v)$, and the curve's tangent, normal, and bi-normal are $\vec{t}_r(t)$, $\vec{n}_r(t)$, and $\vec{b}_r(t)$, respectively. The magnitude of tension along the fiber is $T(t)$. Obviously the direction of tension is $\vec{t}_r(t)$. So $\vec{T_0} = -T(t_0)\vec{t}_r(t_0)$ and $\vec{T_1} = T(t_1)\vec{t}_r(t_1)$. For a small segment of the fiber, the stability condition is

$$-T(t_0)\vec{t}_r(t_0) + T(t_0 + \Delta t)\vec{t}_r(t_0 + \Delta t) + \int_{t_0}^{t_0+\Delta t} \vec{F}_{\text{total}}(t)dt = 0, \tag{3.1}$$

where $\vec{F}_{\text{total}}(t)$ is the total force the surface exerts on the fiber per unit length. $\vec{F}_{\text{total}}(t)$ can be decomposed as

$$\begin{aligned} \vec{F}_{\text{total}}(t) &= N(t)\vec{n}_s(u(t), v(t)) + \vec{fr}(t), \\ \vec{n}_s(u(t), v(t)) \cdot \vec{fr}(t) &= 0, \end{aligned} \tag{3.2}$$

where $N(t)$ is the magnitude of the force in the surface normal direction (support force, $\vec{N}$ in Fig. 3.1), and $fr(t)$ is the magnitude of the remaining force of the support force (friction, $\vec{fr}$ in Fig.

3.1). For simplicity, denote $\vec{n}_s(t) = \vec{n}_s(u(t), v(t))$.

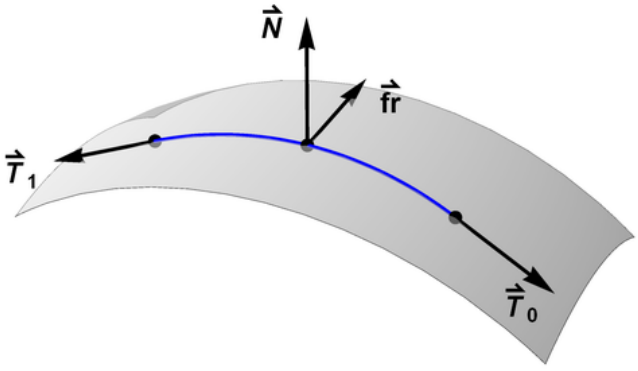


Figure 3.1: Force analysis on a small piece of fiber.

By taking a limit of Δt in Eq. 3.1, we get the derivative form

$$T'(t)\vec{t}_r(t) + T(t)\vec{t_r'}(t) + N(t)\vec{n}_s(t) + \vec{f}r(t) = 0. \tag{3.3}$$

In filament winding, we can always adjust the magnitude of fiber tension by controlling the robot arm. So we suppose that the magnitude of fiber tension is a constant T. It means that the stable condition Eq. 3.3 becomes

$$T\vec{t_r'}(t) + N(t)\vec{n}_s(t) + \vec{f}r(t) = 0. \tag{3.4}$$

For the curve in 3D space,

$$\vec{t_r'}(t) = \|\vec{r'}(t)\|\vec{n}_r(t), \tag{3.5}$$

where $\vec{n}_r(t)$ is the curve's normal. Combining Eq. 3.4 and Eq. 3.5, we have

$$-T\|\vec{r'}(t)\|\vec{n}_r(t) = N(t)\vec{n}_s(t) + \vec{f}r(t). \tag{3.6}$$

As the friction $\vec{fr}$ is always tangent to the surface and the surface normal $\vec{n}_s$ is always perpendicular to the surface, $\vec{n}_s \cdot \vec{fr} = 0$. After squaring both side of Eq. 3.6, we get

$$T^2\|\vec{r'}(t)\|^2 = N(t)^2 + fr(t)^2 + 2N(t)\vec{n}_s \cdot \vec{fr} = N(t)^2 + fr(t)^2. \tag{3.7}$$

Suppose that μ is the coefficient of friction of the surface material (we assume a Coulomb friction model, where frictional force is proportional to normal force). Conceptually, the larger μ is, the rougher the surface is, and thus the more a filament path can deviate from the normal direction without slipping. So, the following inequality is always true when the winding path is stable.

$$fr(t) \leq \mu N(t), \tag{3.8}$$

By plugging in Eq. 3.7 we can get that

$$\begin{aligned} T^2\|\vec{r'}(t)\|^2 &\leq (1+\mu^2)N(t)^2, \\ T\|\vec{r'}(t)\| &\leq \sqrt{1+\mu^2}N(t). \end{aligned} \tag{3.9}$$

T and $\|\vec{r'}(t)\|$ are positive, and $N(t)$ is non-negative. Compute the dot product of Eq. 3.6 and $\vec{n}_s(t)$, then we know

$$T\|\vec{r'}(t)\|(-\vec{n}_r(t) \cdot \vec{n}_s(t)) = N(t). \tag{3.10}$$

Obviously, $-\vec{n}_r(t) \cdot \vec{n}_s(t) \geq 0$. So

$$\begin{aligned} \sqrt{1+\mu^2}N(t)(-\vec{n}_r(t) \cdot \vec{n}_s(t)) &\geq N(t), \\ (-\vec{n}_r(t) \cdot \vec{n}_s(t)) &\geq \frac{1}{\sqrt{1+\mu^2}}, \end{aligned} \tag{3.11}$$

which is equivalent to

$$\mathrm{Angle}(\vec{n}_s(t), -\vec{n}_r(t)) \leq \arctan(\mu), \tag{3.12}$$

where $\mathrm{Angle}(\cdot, \cdot)$ is the angle between two vectors [13].

In Eq. 3.11, we assume that $N(t)$ is positive. When $N(t) = 0$, the winding path must touch the surface without supporting force, which means that the path must be a locally straight line at t. That is

$$\begin{aligned} \vec{n}_r(t) &= 0, \\ \mathrm{Angle}(\vec{n}_s(t), -\vec{n}_r(t)) &= 0 \leq \arctan(\mu). \end{aligned} \tag{3.13}$$

So Eq. 3.12 is still correct.

With no friction (i.e., $\arctan(\mu) = 0$) in Eq. 3.12, we have

$$\vec{n}_s(t) = -\vec{n}_r(t). \tag{3.14}$$

That is, the outward normal of the surface must match the normal of the curve, and thus the path must be geodesic in the absence of friction. For a non-zero μ, the angle between $\vec{n}_s(t)$ and $\vec{n}_r(t)$ must be bounded by $\arctan(\mu)$, as illustrated by Fig. 3.2. The green curve is an exact geodesic and goes through the point $s1$. $\vec{n}_{r1}$ is the curve's normal and $\vec{n}_{s1}$ is the surface normal. Obviously $\vec{n}_{s1} = -\vec{n}_{r1}$ and the angle is 0. The blue curve is an approximate geodesic and goes through the point $s2$. $\vec{n}_{r2}$ is the curve's normal and $\vec{n}_{s2}$ is the surface normal. θ is the angle between $-\vec{n}_{r2}$ and $\vec{n}_{s2}$. For a stable winding path, θ must be no larger than $\arctan(\mu)$.

If we constrain the curve to the surface (the traditional geodesic problem), Eqs. 3.12 and 3.14

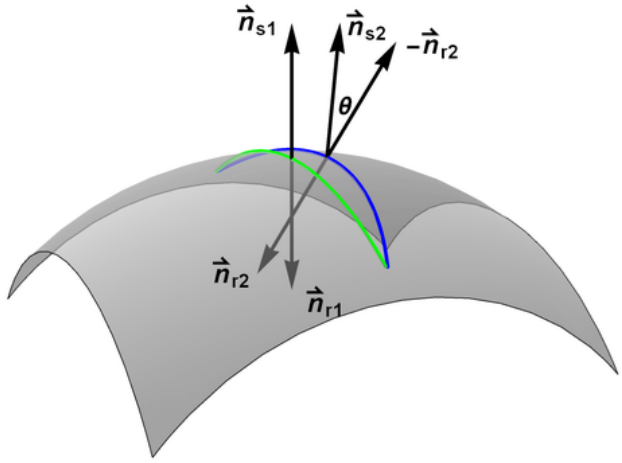


Figure 3.2: Exact geodesic vs. approximate geodesic.

do not hold at concave parts of the surface. In this case, we use

$$\text{Angle}(\vec{n}_s(t), \pm\vec{n}_r(t)) \leq \arctan(\mu), \tag{3.15}$$

$$\vec{n}_s(t) = \pm\vec{n}_r(t). \tag{3.16}$$

Here, $\pm\vec{n}_r(t)$ is determined by the direction of $\vec{n}_r(t)$ and $\vec{n}_s(t)$. When $\vec{n}_r(t) \cdot \vec{n}_s(t) > 0$, we use $\vec{n}_r(t)$ in Eq. 3.15 and Eq. 3.16, and vice versa.

In conclusion, generating a filament winding path solution is equivalent to finding a curve $\vec{r}(t)$ on a given surface $\vec{s}(u, v)$ such that Eq. 3.12 holds.

3.2 Geodesic

Solving for geodesics arises in a number of contexts, and the boundary conditions that are specified vary. But basically, the degrees of freedom (DOFs) of the boundary condition for solving a geodesic equation are 4. Two of the most straightforward conditions are: (a) Provide the start point (2 DOFs), the start direction (a unit vector tangent to the surface at the start point, providing 1 DOF), and the length of the path (1 DOF); or (b) Provide the start point (a point on a 2-manifold, providing 2 DOFs) and the end point (2 DOFs).

Generally speaking, solving geodesic problems with condition (a) is more straightforward than

with condition (b).[1] We start from the given point with given direction, and trace the curve along the curve's tangent or a vector field. We call this a forward solver. However, a forward solution does not work with (b) very well. For condition (b), people usually regard it as a shortest path problem, then solve them iteratively from an initial curve.

In this work, we use condition (b). That is, the two ends of a winding path are always given as the boundary condition. With (a), it is hard to control the final result's patten and we may not generate results valid for manufacturing. In the result part, we show that our method can generate regular winding patterns.

With boundary condition (a), we can use a forward method to solve for the geodesic by starting from the start point and integrating along the tangent direction. This method can generate an accurate geodesic by taking a very small step size. In this work, we will use this method to generate ground truth geodesic. More details about the forward solver are provided in section 3.3.

Using an analytical explicit function to represent the curve can be hard and can have higher complexity when optimizing the curve, so we use a discrete representation of the curve.

$$P_i = \vec{r}(t_i) \quad (i = 1, 2, \ldots, n) \tag{3.17}$$

are a series of points (particles) on the curve. If n is large enough, we can use $\{P_i\}_{i=1}^n$ to approximate the curve. In our algorithm, we optimize the position of $\{P_i\}_{i=1}^n$ iteratively and finally get an approximate result.

3.3 Forward Solver

In this section, we introduce a forward geodesic solver. To simplify the equations, we suppose the parametric curve $\vec{r}(t)$ is parameterized by arc length in this section only. That is,

$$\|\vec{t_r}(t)\| = \|\vec{r'}(t)\| \equiv 1. \tag{3.18}$$

Given a start point and a tangent direction as boundary condition, there is only one geodesic.

[1] (a) is an initial value problem, and (b) is a boundary value problem.

The geodesic to be solved for is $\vec{r}(t)$ (parameterization by arc length). As the surface $\vec{s}(u,v)$ is a parametric surface, $\vec{r}(t)$ on the surface in 3D space has a corresponding curve $(u(t), v(t))$ in the 2D parametric space. Suppose the start point is $P_0 = \vec{r}(t_0) = \vec{s}(u_0, v_0) = \vec{s}(u(t_0), v(t_0))$ and the tangent of the geodesic to be solved for at P_0 is $\vec{t_0} = \vec{t_r}(t_0)$. $\vec{t_0}$ must be a unit vector.

Denote $p(t) = \frac{du(t)}{dt}$ and $q(t) = \frac{dv(t)}{dt}$, $p_0 = p(t_0)$ and $q_0 = q(t_0)$. Then

$$\vec{t_0} = p_0 \frac{\partial \vec{s}}{\partial u}(t_0) + q_0 \frac{\partial \vec{s}}{\partial v}(t_0). \tag{3.19}$$

The differential equation of the geodesic is

$$\begin{cases} \frac{d^2u}{dt^2} + \Gamma^1_{11}\frac{du}{dt}\frac{du}{dt} + 2\Gamma^1_{12}\frac{du}{dt}\frac{dv}{dt} + \Gamma^1_{22}\frac{dv}{dt}\frac{dv}{dt} = 0, \\ \frac{d^2v}{dt^2} + \Gamma^2_{11}\frac{du}{dt}\frac{du}{dt} + 2\Gamma^2_{12}\frac{du}{dt}\frac{dv}{dt} + \Gamma^2_{22}\frac{dv}{dt}\frac{dv}{dt} = 0. \end{cases} \tag{3.20}$$

Γ^k_{ij} are the Christoffel symbols of surface $\vec{s}$:

$$\begin{gathered} \Gamma^1_{11} = \frac{(\vec{s}_v \cdot \vec{s}_v)(\vec{s}_{uu} \cdot \vec{s}_u) - (\vec{s}_u \cdot \vec{s}_v)(\vec{s}_{uu} \cdot \vec{s}_v)}{\|\vec{s}_u \times \vec{s}_v\|^2}, \Gamma^2_{11} = \frac{(\vec{s}_u \cdot \vec{s}_u)(\vec{s}_{uu} \cdot \vec{s}_v) - (\vec{s}_u \cdot \vec{s}_v)(\vec{s}_{uu} \cdot \vec{s}_u)}{\|\vec{s}_u \times \vec{s}_v\|^2}, \\ \Gamma^1_{12} = \frac{(\vec{s}_v \cdot \vec{s}_v)(\vec{s}_{uv} \cdot \vec{s}_u) - (\vec{s}_u \cdot \vec{s}_v)(\vec{s}_{uv} \cdot \vec{s}_v)}{\|\vec{s}_u \times \vec{s}_v\|^2}, \Gamma^2_{12} = \frac{(\vec{s}_u \cdot \vec{s}_u)(\vec{s}_{uv} \cdot \vec{s}_v) - (\vec{s}_u \cdot \vec{s}_v)(\vec{s}_{uv} \cdot \vec{s}_u)}{\|\vec{s}_u \times \vec{s}_v\|^2}, \\ \Gamma^1_{22} = \frac{(\vec{s}_v \cdot \vec{s}_v)(\vec{s}_{vv} \cdot \vec{s}_u) - (\vec{s}_u \cdot \vec{s}_v)(\vec{s}_{vv} \cdot \vec{s}_v)}{\|\vec{s}_u \times \vec{s}_v\|^2}, \Gamma^2_{22} = \frac{(\vec{s}_u \cdot \vec{s}_u)(\vec{s}_{vv} \cdot \vec{s}_v) - (\vec{s}_u \cdot \vec{s}_v)(\vec{s}_{vv} \cdot \vec{s}_u)}{\|\vec{s}_u \times \vec{s}_v\|^2}, \\ \vec{s}_u = \frac{\partial \vec{s}}{\partial u}, \quad \vec{s}_v = \frac{\partial \vec{s}}{\partial v}, \quad \vec{s}_{uu} = \frac{\partial^2 \vec{s}}{\partial u^2}, \quad \vec{s}_{uv} = \frac{\partial^2 \vec{s}}{\partial u \partial v}, \quad \vec{s}_{vv} = \frac{\partial^2 \vec{s}}{\partial v^2}. \end{gathered} \tag{3.21}$$

Replace $\frac{du}{dt}$ and $\frac{dv}{dt}$ with p and q, respectively, then we have:

$$\begin{cases} \frac{du}{dt} = p, \\ \frac{dv}{dt} = q, \\ \frac{dp}{dt} = -\Gamma^1_{11}p^2 - 2\Gamma^1_{12}pq - \Gamma^1_{22}q^2, \\ \frac{dq}{dt} = -\Gamma^2_{11}p^2 - 2\Gamma^2_{12}pq - \Gamma^2_{22}q^2. \end{cases} \tag{3.22}$$

The initial value of the differential equation system is (u_0, v_0, p_0, q_0). (u_0, v_0) is the parametric coordinate of the start point. (p_0, q_0) is the projection of direction $\vec{t_0}$ in the parametric space. p_0 and q_0 can be computed from the initial tangent $\vec{t_0}$ with Eq. 3.19. We use a Runge-Kutta method (RK4) to solve the geodesic in this work.

4. GEODESIC SOLVING

As a winding path must be a geodesic on the surface without friction, the first step of solving the filament winding problem is to find a geodesic on the surface. Our method uses a physics-based model to compute geodesics iteratively.

In filament winding, there may be windings that cross concavities, such as the blue path in Fig. 1.2. Many pure geometric methods do not work in this problem, as they suppose that the curve is always on the surface. Borrowing from physically based simulation, we use a physical method to solve the geodesic problem. In this chapter, we constrain the curve to the surface and formulate the physical method. In Chapter 5, we introduce algorithms to "lift" the curve off the surface.

There are two assumptions that make our physics model ideally suited for our problem. First, we assume that almost all portions of the curve are on the surface. We therefore derive the equations of motion using the 2D parametric surface as the degrees of freedom. Second, we assume that tensile (stretching) and surface contact forces dominate bending and twisting forces. These two assumptions allow us to derive a well-targeted computational model that is more efficient than previous computational models for elastic rod simulation (e.g., [3]).

Table 4.1 shows the notations we use in this dissertation. Our method involves two different spaces: the actual 3D space for winding and the parametric 2D space where the surface is defined. We refer to them as "3D space" and "parametric space," respectively.

In figures in this and the next chapters, unless stated otherwise, a gray curve means an initialization curve; a cyan curve means an exact on-surface geodesic (given by a forward geodesic solver or analytical form); a green curve means an on-surface result given by our algorithm, and a blue curve means an off-surface curve.

4.1 Equations of Motion

A geodesic between two points is a local minimum or maximum curve on the surface. There can be more than one geodesic between two points on a surface, with the curve length being a

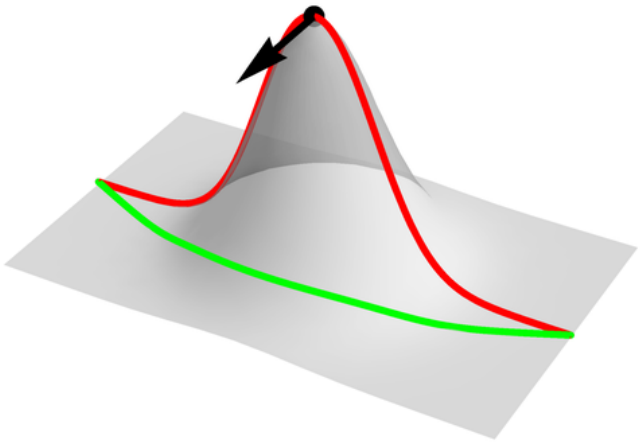


Figure 4.1: An example of geodesic stability. The red curve is a local maximum curve while the green curve is a local minimum curve.

local maximum or a local minimum. In Fig. 4.1, both the red (local maximum) and green (local minimum) curves are geodesics, but the red curve is not stable since it is a local maximum. If the curve is in tension, any small offset (the black arrow) would cause the curve to move to a local minimum (the green curve). Thus, we want to find only locally minimal curves. In our method, we simulate the winding path as particles connected by zero-length springs.

Notation	Meaning
P, V, A	Position, velocity, and acceleration in 3D space
$\overline{P}$, $\overline{V}$, $\overline{A}$	Position, velocity, and acceleration in parametric space
$\mathbf{P}$, $\mathbf{V}$, $\mathbf{A}$	Collection of all particles' position, velocity, and acceleration in 3D space
$\overline{\mathbf{P}}$, $\overline{\mathbf{V}}$, $\overline{\mathbf{A}}$	Collection of all particles' position, velocity, and acceleration in parametric space
J	3 by 2 Jacobian matrix of the mapping function f from parametric space to 3D space
$\mathbf{J}$	Collection of all Jacobian matrices ($3n$ by $2n$)

Table 4.1: Table of notations.

For a particle P_i ($i \notin \{1, n\}$), the force applied to it is

$$F(P_i) = K(P_{i-1} - P_i) + K(P_{i+1} - P_i) = K(I, -2I, I) \cdot (P_{i-1}, P_i, P_{i+1}), \tag{4.1}$$

where K is the spring constant.

Suppose the velocity and acceleration of the particles are $\mathbf{V} = \{V_i\}_{i=1}^n$ and $\mathbf{A} = \{A_i\}_{i=1}^n$ and the mass of all particles are the same, m. P_1 and P_n are given. Then $A_i = F(P_i)/m$, and the collection of 3D accelerations is

$$\frac{d}{dt}\begin{pmatrix}\mathbf{P}\\ \mathbf{V}\end{pmatrix} = \begin{pmatrix}\mathbf{V}\\ \mathbf{A}\end{pmatrix}$$

$$\mathbf{A} = \frac{K}{m}\begin{pmatrix} 0 & & & & & & \\ I & -2I & I & & & & \\ & I & -2I & I & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & I & -2I & I & \\ & & & & & 0 \end{pmatrix}\mathbf{P} = \mathbf{K}\mathbf{P}, \tag{4.2}$$

where I is the 3 by 3 identity matrix. Note that the effect of mass can be handled by appropriately scaling $\mathbf{K}$. (i.e., we assume unit mass.)

Usually, we do physical simulation in the 3D space. In this problem, the winding path mostly stays on the surface during the simulation. If we use general 3D simulation methods, it requires lots of collision detection for each step and this can be very time consuming, as finding the inside and outside needs lots of computation for parametric surfaces. Instead of 3D simulation, we can simulate it in 2D, the parametric space. The motion equation still comes from 3D, but mapped back to 2D with analytical forms. Then we can avoid the collision detection during the simulation. Later in chapter 5, we will describe how we handle curves lifting off at surface concavities. This eliminates costly collision detection calculations.

Suppose the 2D parametric surface is given by a function $\vec{s}(u,v)$. $\mathbf{P} = \{P_i\}_{i=1}^n$ are sampled from the curve. For now, we assume that all particles are on the surface. So $\mathbf{P}$'s corresponding parametric points are $\overline{\mathbf{P}} = \{\overline{P}_i\}_{i=1}^n \in \mathbb{R}^2$ ($P_i = \vec{s}(\overline{P}_i)$). $\overline{\mathbf{V}} = \{\overline{V}_i\}_{i=1}^n$ are the 2D velocities. Then the 3D velocity, V_i, of the particle is

$$\begin{aligned} V_i = \frac{dP_i}{dt} = \frac{dP_i}{d\overline{P}_i}\dot{\overline{P}}_i = J_i\overline{V}_i, \\ J_i = J(\overline{P}_i) = \begin{pmatrix} \vec{s}_u(\overline{P}_i) & \vec{s}_v(\overline{P}_i) \end{pmatrix}, \end{aligned} \tag{4.3}$$

where J_i is the Jacobian matrix at 2D position $\overline{P}_i$, $\overline{V}_i$ is the 2D velocity of $\overline{P}_i$, $\vec{s}_u = \frac{\partial \vec{s}}{\partial u}$, and $\vec{s}_u = \frac{\partial \vec{s}}{\partial u}$. By differentiating Eq. 4.3, we get

$$\begin{aligned} A_i = \dot{V}_i = \dot{J}(\overline{P}_i, \overline{V}_i)\overline{V}_i + J(\overline{P}_i)\dot{\overline{V}}_i, \\ \mathbf{A} = \dot{\mathbf{J}}\overline{\mathbf{V}} + \mathbf{J}\dot{\overline{\mathbf{V}}}, \end{aligned} \tag{4.4}$$

where we use bold letters to indicate global quantities containing all particles. After substituting Eq. 4.2 and rearranging, Eq. 4.4 becomes

$$\begin{aligned} \mathbf{KP} = \dot{\mathbf{J}}\overline{\mathbf{V}} + \mathbf{J}\dot{\overline{\mathbf{V}}}, \\ \mathbf{J}\dot{\overline{\mathbf{V}}} = \mathbf{KP} - \dot{\mathbf{J}}\overline{\mathbf{V}}. \end{aligned} \tag{4.5}$$

$\mathbf{J}$ is a $3n$ by $2n$ matrix and has full column rank as long as the parametric surface is not singular. By multiplying $\mathbf{J}^T$ on both side of the equation, we get

$$\mathbf{J}^T\mathbf{J}\dot{\overline{\mathbf{V}}} = \mathbf{J}^T(\mathbf{KP} - \dot{\mathbf{J}}\overline{\mathbf{V}}). \tag{4.6}$$

Then $\mathbf{J}^T\mathbf{J}$ is a full rank square matrix, and we can solve for the acceleration $\dot{\overline{\mathbf{V}}} = \{\dot{\overline{V}}_i\}_{i=1}^n$.

To make the spring system more stable and converge faster, we consider applying a damping

force to the particles. If a damping force $F_{dp}(P_i) = \beta V_i = \beta J_i \overline{V}_i$ is applied, Eq. 4.5 becomes

$$\mathbf{J}\dot{\overline{\mathbf{V}}} = \mathbf{KP} - \dot{\mathbf{J}}\overline{\mathbf{V}} - \frac{\beta}{m}\mathbf{J}\overline{\mathbf{V}}. \tag{4.7}$$

β is the damping coefficient. The damping force plays a similar role as the friction, which is introduced in a later section. So the damping force can be omitted when the friction is present.

Since we are interested only in the final configuration of the curve, we can safely ignore $\dot{\mathbf{J}}\overline{\mathbf{V}}$, which is 0 when $\overline{\mathbf{V}} = 0$:

$$\mathbf{J}^T\mathbf{J}\dot{\overline{\mathbf{V}}} = \mathbf{J}^T\mathbf{KP}. \tag{4.8}$$

As the computation of $\dot{\mathbf{J}}$ involves $\vec{s}$'s second partial derivatives $\vec{s}_{uu}$, $\vec{s}_{uv}$, and $\vec{s}_{vv}$, ignoring $\dot{\mathbf{J}}\overline{\mathbf{V}}$ can speed up the solving process.

The next step is choosing an integration scheme for the system. We discuss different schemes below. We will use the following notation:

Given an integration variable Z (Z can be P, V, A, $\overline{P}$, $\overline{V}$, J, etc.), Z_i represents the variable for the corresponding particle P_i. The time step is $\Delta t = h$. $Z^{(k)}$ means Z after the k-th step. For example, $V_i^{(k)}$ means particle P_i's 3D velocity after the k-th step and $\mathbf{V}^{(k)}$ consists of all the $V_i^{(k)}$ after the k-th step ($t = t_0 + kh$).

4.2 Time Integration of Explicit Method

Euler. The most straightforward integration uses Euler's (i.e., a first order) method. Applying Euler integration to Eq. 4.6, we have

$$\mathbf{J}^{(k)T}\mathbf{J}^{(k)}(\overline{\mathbf{V}}^{(k+1)} - \overline{\mathbf{V}}^{(k)}) = h\mathbf{J}^{(k)T}(\mathbf{KP}^{(k)} - \dot{\mathbf{J}}^{(k)}\overline{\mathbf{V}}^{(k)}). \tag{4.9}$$

The equations for an explicit Euler method are

$$\begin{aligned} \overline{\mathbf{V}}^{(k+1)} &= \overline{\mathbf{V}}^{(k)} + h(\mathbf{J}^{(k)T}\mathbf{J}^{(k)})^{-1}\mathbf{J}^{(k)T}(\mathbf{K}\mathbf{P}^{(k)} - \dot{\mathbf{J}}^{(k)}\overline{\mathbf{V}}^{(k)}), \\ \overline{\mathbf{P}}^{(k+1)} &= \overline{\mathbf{P}}^{(k)} + h\overline{\mathbf{V}}^{(k)}. \end{aligned} \tag{4.10}$$

Applying a damping force, the equation for 2D velocity becomes:

$$\overline{\mathbf{V}}^{(k+1)} = \overline{\mathbf{V}}^{(k)} + h(\mathbf{J}^{(k)T}\mathbf{J}^{(k)})^{-1}\mathbf{J}^{(k)T}(\mathbf{K}\mathbf{P}^{(k)} - \dot{\mathbf{J}}^{(k)}\overline{\mathbf{V}}^{(k)}) - h\frac{\beta}{m}\overline{\mathbf{V}}^{(k)}. \tag{4.11}$$

We can also derive a simpler version from Eq. 4.8:

$$\overline{\mathbf{V}}^{(k+1)} = \overline{\mathbf{V}}^{(k)} + h(\mathbf{J}^{(k)T}\mathbf{J}^{(k)})^{-1}\mathbf{J}^{(k)T}\mathbf{K}\mathbf{P}^{(k)}. \tag{4.12}$$

Simple Euler integration is very unstable in this situation and exploded in several steps, even with damping, in our experiments.

Runge-Kutta. Runge-Kutta methods have higher orders than the Euler Method. As Runge-Kutta methods are also explicit, they are still not stable enough. In experiments, we used RK4 and RK4 is more stable than the explicit Euler method. But the result still exploded in several steps.

4.3 Time Integration of Implicit Method

From the explicit methods we discussed above, we find out that the explicit methods are unstable in the geodesic problem. Thus, we turn to implicit integration for stability.

Using backward Euler integration, we use the acceleration from step $k+1$ in Eq. 4.8 (Eq. 4.6 without $\dot{\mathbf{J}}\overline{\mathbf{V}}$), so it becomes

$$\mathbf{J}^{(k+1)T}\mathbf{J}^{(k+1)}(\overline{\mathbf{V}}^{(k+1)} - \overline{\mathbf{V}}^{(k)}) = h\mathbf{J}^{(k+1)T}\mathbf{K}\mathbf{P}^{(k+1)}. \tag{4.13}$$

Given the nonlinear nature of $\mathbf{J}^{(k+1)}$, we cannot have an analytical equation for $\overline{\mathbf{V}}^{(k+1)}$, so we

approximate $\mathbf{J}^{(k+1)}$ with $\mathbf{J}^{(k)}$. The positions, $\mathbf{P}^{(k+1)}$ are approximated with backward Euler as

$$\begin{aligned}\mathbf{P}^{(k+1)} =& \vec{s}(\overline{\mathbf{P}}^{(k+1)}) \\ =& \vec{s}(\overline{\mathbf{P}}^{(k)} + h\overline{\mathbf{V}}^{(k+1)}) \\ \approx& \vec{s}(\overline{\mathbf{P}}^{(k)}) + h\mathbf{J}^{(k+1)}\overline{\mathbf{V}}^{(k)} \\ =& \mathbf{P}^{(k)} + h\mathbf{J}^{(k+1)}\overline{\mathbf{V}}^{(k)}.\end{aligned} \tag{4.14}$$

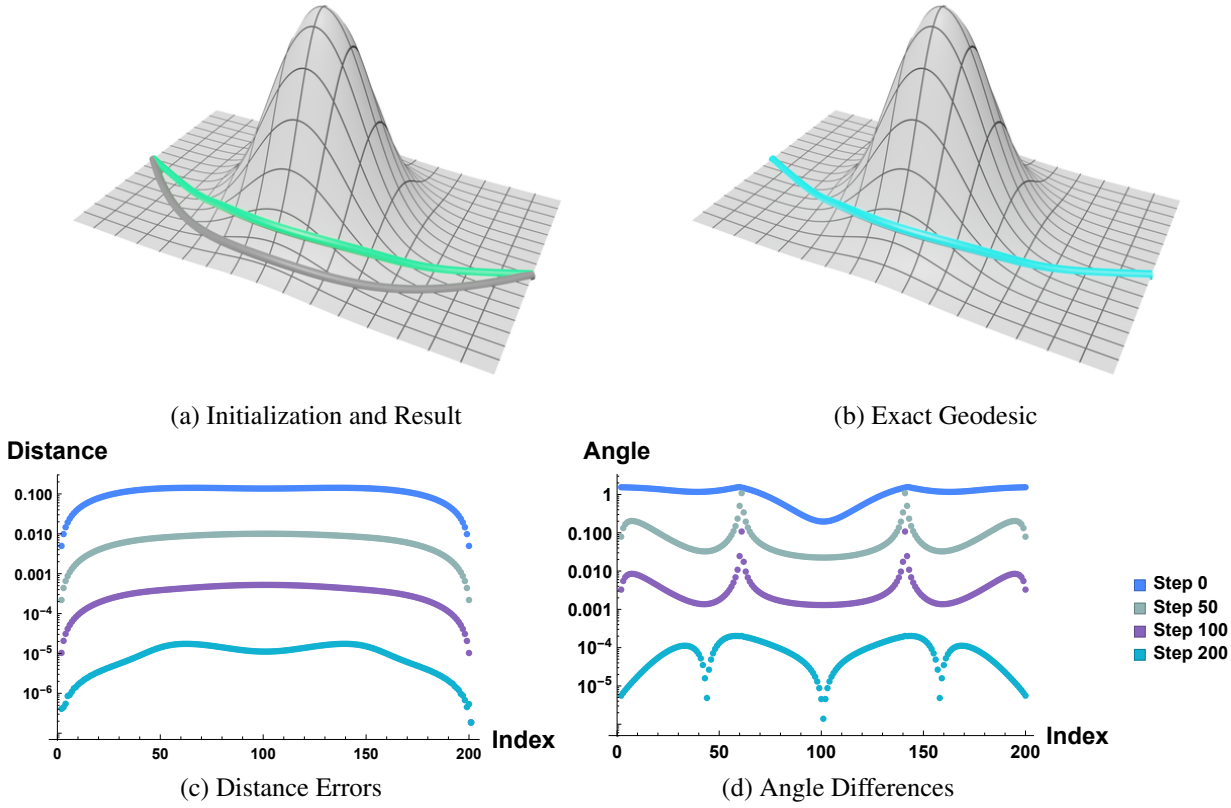
Making these substitutions for $\mathbf{P}^{(k+1)}$ and $\mathbf{J}^{(k+1)}$, and dropping the superscript (k) in Eq. 4.13, we get

$$\mathbf{J}^T\mathbf{J}(\overline{\mathbf{V}}^{(k+1)} - \overline{\mathbf{V}}) = h\mathbf{J}^T\mathbf{K}(\mathbf{P} + h\mathbf{J}\overline{\mathbf{V}}^{(k+1)}). \tag{4.15}$$

Arranging the terms so that $\overline{\mathbf{V}}^{(k+1)}$ is on the left hand side, we get

$$\begin{aligned}\mathbf{J}^T(\mathbf{I} - h^2\mathbf{K})\mathbf{J}\overline{\mathbf{V}}^{(k+1)} &= \mathbf{J}^T\mathbf{J}\overline{\mathbf{V}} + h\mathbf{J}^T\mathbf{K}\mathbf{P}, \\ \overline{\mathbf{P}}^{(k+1)} &= \overline{\mathbf{P}} + h\overline{\mathbf{V}}^{(k+1)},\end{aligned} \tag{4.16}$$

where $\mathbf{I}$ is the $3n \times 3n$ identity matrix. We solve this linear system at every time step for the parametric velocities, $\overline{\mathbf{V}}^{(k+1)}$, which are then used to update the parametric positions, $\overline{\mathbf{P}}^{(k+1)}$. Fig. 4.2 shows an example of our method demonstrating that it produces good convergence and accuracy. In (a), the gray curve shows a path with two fixed ends on a surface. Using the gray curve as an initialization, our method can generate a geodesic (the green curve). As a reference, we also show an exact geodesic sharing the same end points in (b). Our result and the exact geodesic are visually identical. We measure the errors of all particles from our method's result path and plot the errors in (c) and (d). We measure the errors after 50, 100, 150 and 200 steps. (c) shows the distance errors. For each particle on the curve, we compute the minimal distance from the particle to the exact geodesic. (d) shows the angle differences described in Eq. 3.12. According to the conclusion, the angle difference between the surface normal $\vec{n}_s$ and the opposite curve normal $-\vec{n}_r$



(a) Initialization and Result

(b) Exact Geodesic

(c) Distance Errors

(d) Angle Differences

Figure 4.2: An example of our simulation result. In (a), the gray curve is the initialization curve, and the green curve is the result after 50 iterations. As a comparison, the cyan curve in (b) shows the exact result, which is visually the same as our result. (c) **LOG** plot that shows the distance from each particle to the exact curve after different numbers of simulation steps. The horizontal axis is the index of each particle and the vertical axis is distance. As a reference, the distance between the two curve ends (Point A and B) is 2. (d) **LOG** plot that shows the angle difference between $\vec{n}_s$ and $-\vec{n}_r$ described in Eq. 3.12.

should be 0 if the curve is an exact geodesic. A small angle difference indicates that the curve is close to an exact geodesic. The result shows that our method converge to an exact geodesic rapidly.

With a damping force, which is optional, Eq. 4.16 becomes

$$\begin{aligned} \mathbf{J}^T((1+h\frac{\beta}{m})\mathbf{I}-h^2\mathbf{K})\mathbf{J}\overline{\mathbf{V}}^{(k+1)} &= \mathbf{J}^T\mathbf{J}\overline{\mathbf{V}}+h\mathbf{J}^T\mathbf{K}\mathbf{P}, \\ \overline{\mathbf{P}}^{(k+1)} &= \overline{\mathbf{P}}+h\overline{\mathbf{V}}^{(k+1)}, \end{aligned} \tag{4.17}$$

4.4 Friction

To this point, we have not taken friction into consideration in our physically based simulation. The method discussed above constrains the curve to be on a smooth surface. But in real-world manufacturing, friction also needs to be considered. With friction, the winding path doesn't have to be an exact geodesic.

Given friction coefficient μ, we can use Eq. 3.12 as a stopping criterion. However, this does not allow friction to influence the motion of particles during the simulation. For different μ, the only difference is when to stop. So, we simulate the frictional force in each step. If the motion of the path has stopped and all forces are balanced, Eq. 3.12 must hold for the current path.

We extend the frictional impulse approach of Bridson et al. [4]. In their original formulation, the velocity of each particle is decomposed into a component tangential and a component perpendicular to the surface, and then the tangential velocity is updated using the coefficient of friction μ. With our method, the particles are on the surface, and the particle velocity is automatically tangential to the surface, and so we can directly modify the velocity.

Let us denote the pre-friction velocity (output of Eq. 4.16) as $\overline{V}_i^{\text{pre}}$ in 2D and V_i^{pre} in 3D. Then update $V_i^{(k+1)}$ with

$$V_i^{(k+1)} = \max(1 - \frac{h\mu A_i^{(k)} \cdot \vec{n}_s(P_i^{(k)})}{\|V_i^{(k+1)\text{pre}}\|}, 0) V_i^{(k+1)\text{pre}}, \tag{4.18}$$

where $\vec{n}_s(\cdot)$ is the surface normal. $A_i^{(k)}$ is the 3D acceleration which can be computed from the spring force. Because $V_i^{(k+1)\text{pre}} = J_i^{(k)} \overline{V}_i^{(k+1)\text{pre}}$ and $V_i^{(k+1)} = J_i^{(k)} \overline{V}_i^{(k+1)}$, we multiply $(J_i^{(k)T} J_i^{(k)})^{-1} J_i^{(k)T}$ on both sides of the equation. Then $\overline{V}_i^{(k+1)}$ can be updated by

$$\overline{V}_i^{(k+1)} = \max\left(1 - \frac{h\mu |A_i \cdot \vec{n}_s(P_i)|}{\|V_i^{\text{pre}}\|}, 0\right) \overline{V}_i^{\text{pre}}, \tag{4.19}$$

which is our final, post-friction velocity.

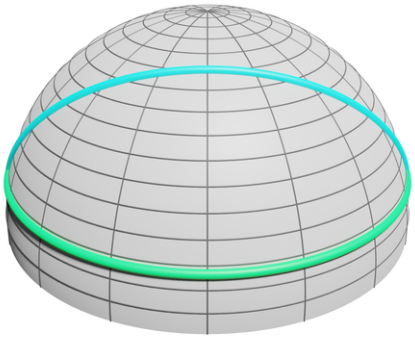
Fig. 4.3 shows identical experiments varying only μ. All setups are the same except μ in Fig.

4.3-a to Fig. 4.3-d. The gray curve is the initial curve, the cyan curve is ground truth, and the green curves are simulation results.

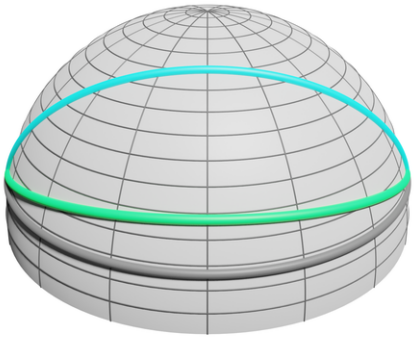
In Fig. 4.3-a, friction is so large that the curve stays at the initial position and does not move. If we decrease μ, friction becomes smaller and the curve moves closer to the geodesic (shown in Fig. 4.3-b and Fig. 4.3-c). If μ is too small, the curve will converge to the geodesic (Fig. 4.3-d). In Fig. 4.3-g, we compute the mean angle differences between $\vec{n}_s$ and $\vec{n}_r$ at all locations in a single step. The plot shows that the mean difference converges to (or, in some cases, becomes smaller than) $\arctan(\mu)$. The convergence shows that the curve reaches a stable solution satisfying Eq. 3.12. In the four experiments, we also include a non-zero damping force.

Fig. 4.4 shows a more non-symmetric model example with different friction coefficients. Generally speaking, a larger friction coefficient, which means rougher surface, can generate more stable winding paths. In this example, the heel part can be covered much better when the friction coefficient is large.

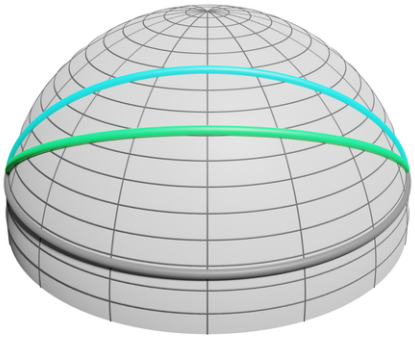
We can apply variable coefficient of friction on the surface. Fig. 4.5 shows an example. The dark gray area in (a) shows a larger friction area ($\mu = 0.4$) while the light gray area represents a smaller friction ($\mu = 0.1$). Our method shows that the path stays on the convex area due to the larger friction.



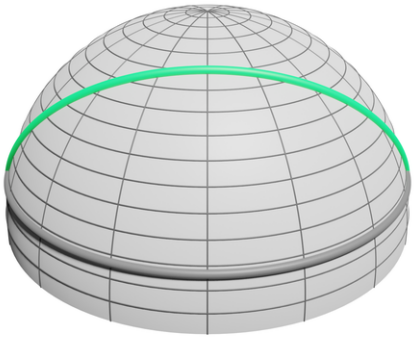
(a) $\mu = 0.3$



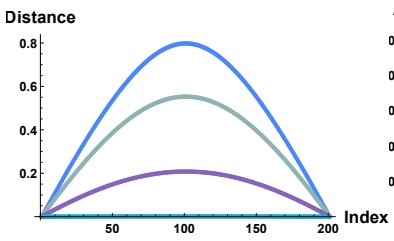
(b) $\mu = 0.2$



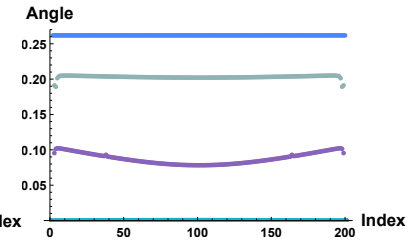
(c) $\mu = 0.1$



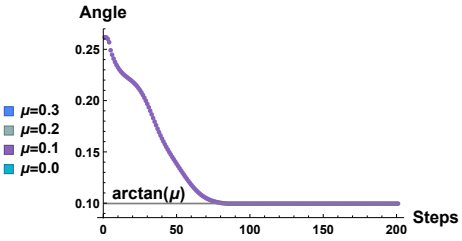
(d) $\mu = 0.0$



(e) Distance Differences



(f) Angle Differences

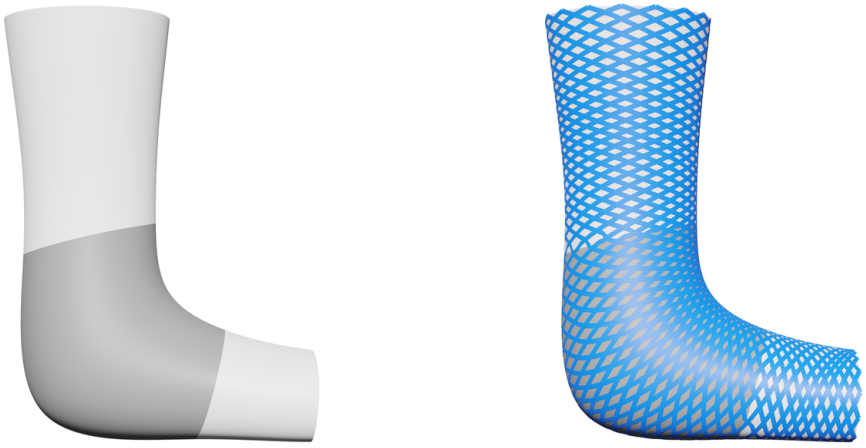


(g) Mean Angle Differences ($\mu = 0.1$)

Figure 4.3: An example of how friction affects the simulation. (a)-(d) show four different experiments with varying μ values. The surface is a parametric semi-sphere. The initial path is gray, and the analytically calculated exact geodesic is cyan. The green curves are the simulation results after 200 iterations with different μ. From (a) to (d), μ is 0.3, 0.2, 0.1, and 0.0, respectively. Similar to Fig. 4.2, (e) and (f) show the difference from the geodesic. (e) shows the distance from the result to the exact geodesic, and (f) shows the angle difference between $\vec{n}_s$ and $-\vec{n}_r$. In (g), we averaged all angle differences in a single step for $\mu = 0.1$. The horizontal axis is the time step. The vertical axis is the mean angle difference. We also marked $\arctan(\mu)$ mentioned in Eq. 3.12 in plot (g).

(a) $\mu = 0.05$ (b) $\mu = 0.2$ (c) $\mu = 0.4$

Figure 4.4: Leg cast model with different friction coefficients. Higher friction makes fibers harder to slide.



(a) (b)

Figure 4.5: Leg cast model with variable friction coefficient. In (a), the light gray area's friction coefficient is $\mu = 0.1$ and the dark gray area's friction coefficient is $\mu = 0.4$.

5. WINDING PATH GENERATING

In the previous chapter, our method constrains the curve to be on a surface. Multiplying $\mathbf{J}^T$ on both sides of Eq. 4.4 mapped the force to the tangent plane at each particle. While this works well for finding a geodesic on the surface (or, with friction, an approximate geodesic), we need additional techniques to handle fibers lifting off of and landing onto the surface during simulation. Again, we take advantage of the fact that we work in the reduced, parametric space. First, in section 5.1, we describe how we detect concavities. Then in section 5.2, we describe how we modify the equations of motion to handle lifted points. Finally in section 5.3, we describe how we utilize a one-way coupled simulation to keep track of the lifted points to speed up collision detection.

5.1 Lifting From the Surface

To allow a particle to leave the surface, a concavity detection and lifting algorithm is necessary. In this part, we use Fig. 5.1 to demonstrate how our lifting algorithm works. Fig. 5.1-a shows a path with fixed ends on a non-convex surface. The blue curve in Fig. 5.1-b is the ideal solution.

If the curve normal and the surface normal are in the same direction,

$$n_s \cdot n_r > 0, \tag{5.1}$$

the force will move the fiber off the surface. We can calculate a discrete normal of the curve as $P_{i-1} - 2P_i + P_{i+1}$. We mark particles in this state as "off-surface" (and otherwise call them "on-surface"). On-surface particles are connected with the nearest previous and next subsequent on-surface particles. We use P_{i^-} and P_{i^+} to represent the previous and next on-surface particles of P_i. In Fig. 5.2, when $i = 2$, then $P_{2^-} = P_1$ and $P_{2^+} = P_5$. When $i = 5$, then $P_{5^-} = P_2$ and $P_{5^+} = P_6$.

The green curve in Fig. 5.1-a represents a path consisting of only particles on convex parts. All other concave parts are marked as blue. Fig. 5.1-c is a side-view of the curve and surface.

In many cases, the curve is locally convex, but still needs to leave the surface. In Fig. 5.1-c, if

we mark all points on locally concave regions as off-surface and lift them, the curve will become as in Fig. 5.1-d. Obviously, the curve is still not stable, and current on-surface particles will leave the surface due to the fiber tension.

To model this behavior, we repeatedly calculate the new curve normal, using P_{i^-} and P_{i^+} (rather than P_{i-1} and P_{i+1}): $(P_{i^-} - 2P_i + P_{i^+})$. This may result in more points being lifted, requiring repeated normal reevaluation, but the process will end.

If we apply this process to Fig. 5.1-d, we will get an updated path similar to that shown in Fig. 5.1-e. Continuing the process until no new particles leave the surface results in Fig. 5.1-f. Algo. 1 describes the lifting algorithm (which could be optimized further).

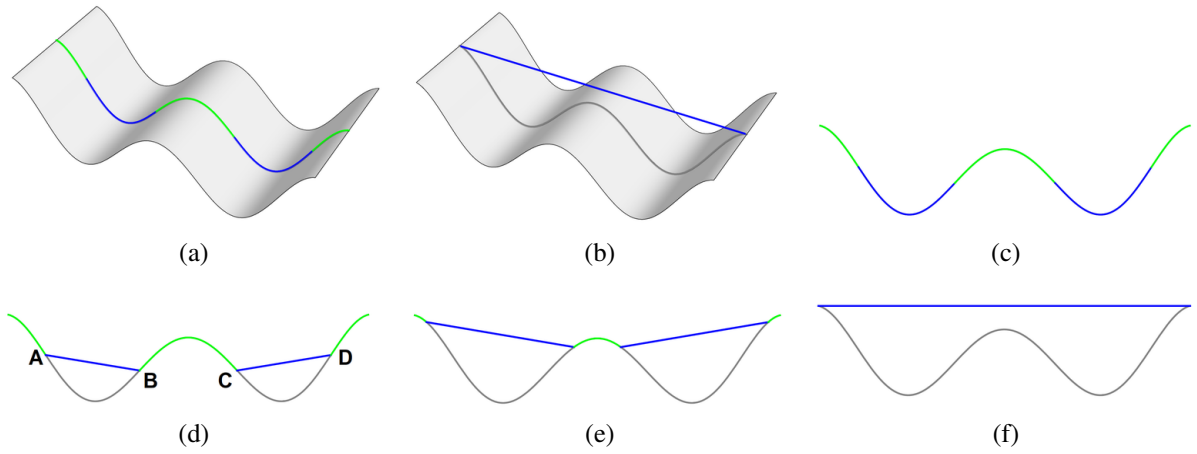


Figure 5.1: Lifting algorithm. (a) shows an initial on-surface curve, while the result should be as seen in (b). Blue means locally concave and green means locally convex. (c) is the side view of the curve. If we lift all the particles at locally concave regions, it will generate a path like (d). (e) shows application of our algorithm to (d) to lift more particles, finally resulting in (f).

5.2 Curves with Off-Surface Points

In filament winding, the off-surface part of the fiber is always a straight line when stable or moving relatively slowly. So we assume that all the off-surface particles (the blue particles in Fig. 5.2-a) are distributed evenly on the straight line between two on-surface particles. With this

ALGORITHM 1: Lifting Algorithm

Data: Points on the curve $\{P_i\}$, surface normal on these points $\{N_i\}$
Result: A boolean array $\{\text{on\_surface}_i\}$ indicates whether each point is still on the surface
Set $\text{detected\_leaving} = \text{True}$;
while detected_leaving **do**
    $\text{detected\_leaving} = \text{False}$;
    for *Each on-surface particle* P_i **do**
        Find the first on-surface particle before it and after it, P_{i^-} and P_{i^+};
        if $(P_{i^-} - 2P_i + P_{i^+}) \cdot N_i > 0$ **then**
            Mark P_i as off-surface: $\text{on\_surface}_i = \text{False}$;
            Update P_i's 3D location with the linear interpolation of P_{i^+} and P_{i^-};
            $\text{detected\_leaving} = \text{True}$
        end
    end
end

assumption, we ignore the regular motion and vibration in the 3D space, which are not important for the filament winding problem. These particles won't affect the motion of on-surface particles.

We will show how to modify the current system (Eq. 4.6) in the next section.

(a) (b)

Figure 5.2: Off-surface particles.

Assume that the subset of k on-surface particles are given, in order, by $P_{i_1}, P_{i_2}, \ldots, P_{i_k} (1 = i_1 < i_2 < \cdots < i_k = n)$. We can ignore all off-surface particles (in between some of those on-surface particles) and set the spring force using just on-surface particles. So, the total force on

P_{i_j} is just $K(\frac{1}{i_j - i_{j-1}} P_{i_{j-1}} - \frac{i_{j+1}-i_{j-1}}{(i_j - i_{j-1})(i_{j+1}-i_j)} P_{i_j} + \frac{1}{i_{j+1}-i_j} P_{i_{j+1}})$ So the matrix $\mathbf{K}$ becomes

$$\tilde{\mathbf{A}} = \frac{K}{m} \begin{pmatrix} \mathbf{0} & & & & \\ a_2\mathbf{I} & b_2\mathbf{I} & a_3\mathbf{I} & & \\ & \ddots & \ddots & \ddots & \\ & & a_{k-1}\mathbf{I} & b_{k-1}\mathbf{I} & a_k\mathbf{I} \\ & & & & \mathbf{0} \end{pmatrix} \tilde{\mathbf{P}} = \tilde{\mathbf{K}}\tilde{\mathbf{P}}, \tag{5.2}$$

$$a_j = \frac{1}{i_j - i_{j-1}}, \quad b_j = -\frac{i_{j+1} - i_{j-1}}{(i_j - i_{j-1})(i_{j+1} - i_j)}.$$

$\tilde{\mathbf{A}}$ and $\tilde{\mathbf{P}}$ mean that only the on-surface particles are involved. $\tilde{\mathbf{K}}$ is a smaller matrix than $\mathbf{K}$.

As an example, suppose there are 6 total particles in Fig. 5.2-b. P_1 and P_6 are endpoints. P_2 and P_5 are on-surface, while P_3 and P_4 are off-surface. In this case, $\tilde{\mathbf{A}}$ consists of only 4 rows that correspond to P_1, P_2, P_5, and P_6, respectively. P_2 and P_5 are connected, and the forces applied to P_2 are $K_s\overrightarrow{P_2P_1}$ and $\frac{K_s}{3}\overrightarrow{P_2P_5}$ (as P_2 and P_5 are connected by $5 - 2 = 3$ springs). P_5 is similar. So the $\tilde{\mathbf{A}}$ in this example is:

$$\tilde{\mathbf{A}} = \frac{K}{m} \begin{pmatrix} \mathbf{0} & & & \\ \mathbf{I} & -\frac{4}{3}\mathbf{I} & \frac{1}{3}\mathbf{I} & \\ & \frac{1}{3}\mathbf{I} & -\frac{4}{3}\mathbf{I} & \mathbf{I} \\ & & & \mathbf{0} \end{pmatrix} \tilde{\mathbf{P}}. \tag{5.3}$$

5.3 Landing On the Surface

Off-surface particles might return to (land on) the surface in later iterations. Rather than applying separate collision detection for particles, we present here a modification to Eq. 4.6 to handle this problem. Algo. 2 describes our process.

Recall that off-surface particles are not simulated in 3D; we just interpolate their positions from on-surface particles. For each off-surface particle, we first find the closest point on the surface. We

ALGORITHM 2: Landing Algorithm

Data: Points on the curve $\{P_i\}$, the parametric surface f, a boolean array $\{on\_surface_i\}$ described in Algo. 1, an approximate result $\{\overline{P}_i\}$

Result: Updated $\{on\_surface_i\}$ and the mapping back result $\{\overline{P}'_i\}$

for *Each off-surface particle* P_i **do**

  Use Newton method to find out the closest point on the surface $P'_i = \vec{s}(\overline{P}'_i)$;

  Compute the surface normal N_i at P'_i;

  if $(P_i - P'_i) \cdot N_i \leq 0$ **then**

    Mark P_i as on-surface: $on\_surface_i = True$;

    Update P_i with P'_i

  end

end

use Newton's method to find this point. The objective function is

$$g(\overline{P}) = \|\vec{s}(\overline{P}) - P\|^2. \tag{5.4}$$

Then $\overline{P}$ can be updated by

$$\overline{P}^{(k+1)} = \overline{P}^{(k)} - H_g(\overline{P}^{(k)})^{-1}\nabla g(\overline{P}^{(k)}). \tag{5.5}$$

H_g is g's (2 by 2) Hessian matrix and ∇g is g's gradient vector.

$$\begin{cases} (H_g)_{11} = \vec{s}_{uu} \cdot (\vec{s} - P) + \|\vec{s}_u\|^2, \\ (H_g)_{12} = (H_g)_{21} = \vec{s}_{uv} \cdot (\vec{s} - P) + \vec{s}_u \cdot \vec{s}_v, \\ (H_g)_{22} = \vec{s}_{vv} \cdot (\vec{s} - P) + \|\vec{s}_v\|^2, \end{cases} \tag{5.6}$$

$$\nabla g = 2J_{\vec{s}}^T(\vec{s} - P). \tag{5.7}$$

Newton's method is fast but sensitive to the initial value and can converge to different solutions. In the landing algorithm, we need an initial guess for the nearest point for each particle and the final result could be far away with a bad initialization. So, a reasonable initial guess for Newton's

method is important.

In the previous formulation, we ignore all motions of off-surface particles during simulation. Now, we will add back points on the surface corresponding to off-surface points. The actual positions of off-surface particles are still in the straight line defined by on-surface particles, while we simulate a copy of these particles on the surface at the same time. These false particles are like the "shadow" of off-surface particles on the surface, so we call them "shadow particles" and the resulting curve the "shadow path". We then use the shadow path to initialize the Newton solver.

If we simply use Eq. 4.2, the on-surface particles will be influenced by the shadow particles, which is not desirable. So a possible solution is to combine Eq. 4.2 and Eq. 5.2, and replace the rows corresponding to on-surface particles in $\mathbf{K}$ with the equivalent rows in $\tilde{\mathbf{K}}$. To be specific, when filling the i-th row (in 3 by 3 blocks, $2 \leq i \leq n-1$) of $\mathbf{K}$, if P_i is on-surface and $i = i_j$, then

$$\begin{aligned} \mathbf{K}_{ii} &= -\frac{i_{j+1} - i_{j-1}}{(i - i_{j-1})(i_{j-1} - i)} \frac{K}{m} \mathbf{I}, \\ \mathbf{K}_{ii_{j-1}} = \frac{1}{i - i_{j-1}} \frac{K}{m} \mathbf{I}, &\quad \mathbf{K}_{ii_{j+1}} = \frac{1}{i_{j+1} - i} \frac{K}{m} \mathbf{I}. \end{aligned} \tag{5.8}$$

Otherwise,

$$\mathbf{K}_{ii} = -2\frac{K}{m}\mathbf{I}, \quad \mathbf{K}_{ii-1} = \mathbf{K}_{ii+1} = \frac{K}{m}\mathbf{I}. \tag{5.9}$$

All other elements are filled with zeros. Then all on-surface particles are still moved as expected while the shadow path is influenced by the on-surface particles.

In the concrete example mentioned in section 5.1, the motion equation becomes

$$\tilde{\mathbf{A}}' = \frac{K}{m}\begin{pmatrix} \mathbf{0} & & & & & \\ \mathbf{I} & -\frac{4}{3}\mathbf{I} & & & \frac{1}{3}\mathbf{I} & \\ & \mathbf{I} & -2\mathbf{I} & \mathbf{I} & & \\ & & \mathbf{I} & -2\mathbf{I} & \mathbf{I} & \\ & \frac{1}{3}\mathbf{I} & & & -\frac{4}{3}\mathbf{I} & \mathbf{I} \\ & & & & & \mathbf{0} \end{pmatrix} \tilde{\mathbf{P}}', \tag{5.10}$$

where $\tilde{\mathbf{P}}'$ consists of P_1, P_2, P_3', P_4', P_5, and P_6.

Eq. 5.10 is a size $3n$ by $3n$ linear system. But, as the on-surface parts and shadow paths are one-way coupled, both the motion equations and results for on-surface particles should be the same as Eq. 5.2. So, we can solve the on-surface particles with Eq. 5.2 first, then solve for other shadow path particles with Eq. 5.10. The system is decomposed into two smaller systems, and can be solved faster than solving Eq. 5.10 directly.

Once we have P_i', the nearest point to P_i, we can check whether a collision occurred by determining whether P_i has moved below the surface. This can be done by forming the vector $P_i - P_i'$ and comparing direction to the surface normal at P_i', N_i. If the direction is opposite, a collision has happened, and we can mark that point as an on-surface point, with position P_i'.

If the distance from the real particle to the closest point is smaller than a given threshold, it means that a collision may have happened. And if the direction from the particle to its closest point is same as the face normal, that means that a collision happened and the particle has landed on the surface. Algo. 2 describes our landing process.

Fig. 7.3-(a-c) shows an example of off-surface simulation.

6. END-TO-END SYSTEM AND WINDING PATH EVALUATION

With our solver, winding paths can be generated easily. In this section, we will demonstrate an end-to-end system for filament winding design.

Fig. 6.1 shows the workflow of our system. First a user designs a shape as input. The user can also specify winding parameters at the same time. Then the path can be generated with our solver. Our solver also provides an analysis of the winding path to guide the user to edit the surface. Users can repeat the edit-generate-analyze loop until they are satisfied with the winding path.

In our implementation, we use cubic B-spline surfaces, but our end-to-end approach works with any kind of parametric surface. We focus on the parametric surfaces that are homotopic to a cylinder: $\vec{s}(u, v) = \vec{s}(u + T_u, v)$, where T_u is the minimum positive period with any v.

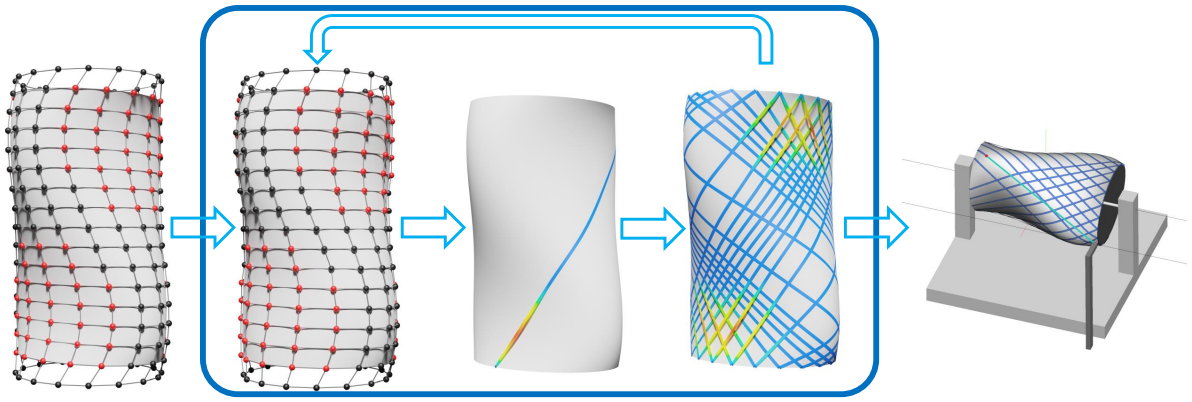
6.1 Winding Parameters

As described in the previous section, our solver uses a number of parameters, including the particle count, spring stiffness, and friction coefficient. Experimentally we have found that the default parameter values can generate geodesics successfully in most cases and do not influence the result significantly. Another parameter, the number of curves generated, will influence the accuracy with which the analysis (see section 6.2) is performed. The parameter that most influences the results is the number of revolutions of the curves. In the rest of this section, we describe an effective method for automatically computing this parameter.

As our end-to-end system focuses on shapes homotopic to a cylinder without two ends, the number of revolutions can be defined as

$$n_{rev} = \frac{u_n - u_1}{T_u}, \tag{6.1}$$

where $P_1 = (u_1, v_1)$ and $P_n = (u_n, v_n)$ are the two ends of a winding path. v_1 and v_n correspond to the top and bottom v values. Once the number of curves is determined, we assume that the end points of all paths are distributed evenly on the two ends of the cylinder. In the next section, we



(a) Surface Design (b) User Editing (c) Path Generation (d) Visualization (e) Manufacturing

Figure 6.1: Workflow of our end-to-end filament winding system. The input of our system is a parametric surface (a) and winding parameters. (b)-(d) show that users can design and evaluate winding paths over several iterations. (e) shows a simulated winding process with the generated paths from (c).

will show a non-uniformly distributed paths setup.

Since the endpoints of our curves do not change as we determine the curve path, the number of revolutions is fixed by this parameter. And, since the geodesic curve is determined based on the starting and ending points, this number of revolutions will have a major influence on the final path.

An inappropriate number of revolutions may lead to off-surface winding paths, which are not desirable. Generally speaking, a better initialization may create better results when the path does not move a lot during the simulation. Fig. 6.2 shows results with different numbers of revolutions.

As the initial path is a straight line in parametric space, the initial path also depends on the number of revolutions. The initial path is

$$v = \frac{v_n - v_1}{T_u n_{rev}}(u - u_1) + v_1. \tag{6.2}$$

If the initial path is always on the surface, the result may have fewer parts lifted. If a point (u_0, v_0) is on an initial path, the path is on the surface at this point when the normal curvature in the

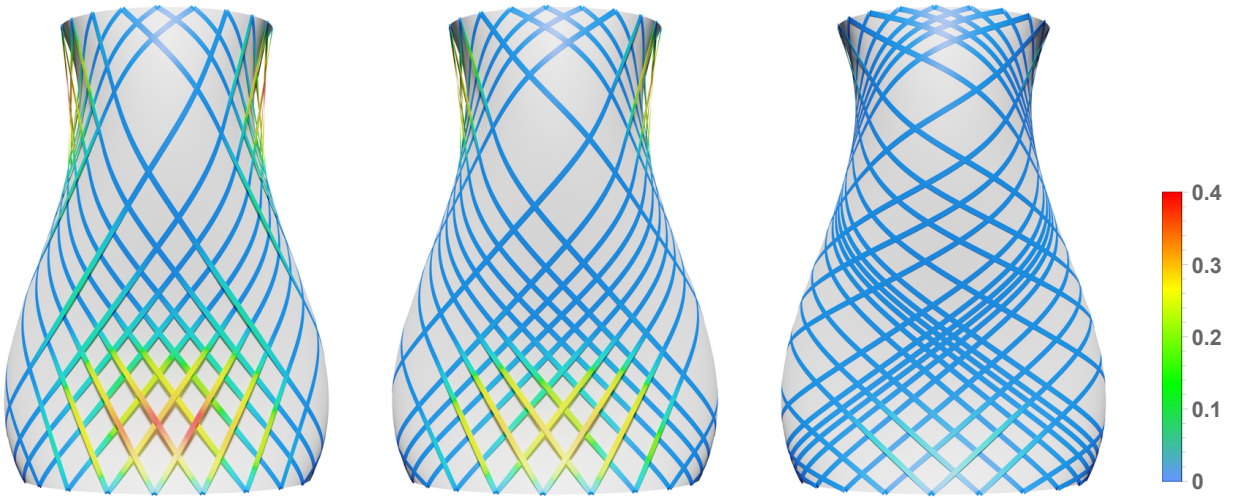


Figure 6.2: Different numbers of revolutions will lead to significantly different results. From left to right, the numbers of revolutions are 0.4, 0.5, and 0.8, respectively. The color shows the distance from the winding path to the surface. All other parameters are the same.

direction $(1, \frac{v_n - v_1}{T_u n_{rev}})$ points out of the surface:

$$(\vec{s}_{uu} + \frac{v_n - v_1}{T_u n_{rev}} \vec{s}_{uv} + (\frac{v_n - v_1}{T_u n_{rev}})^2 \vec{s}_{vv}) \cdot (\vec{s}_u \times \vec{s}_v) > 0. \tag{6.3}$$

After simplification, this is a quadratic function of n_{rev}, and the solution consists of zero, one, or two intervals.

We find all the intervals of n_{rev} for all the points on the surface, and compute the most common interval(s) of these intervals. We pick an initial value from the most common interval(s), so that Eq. 6.3 holds for all or most of the points on the surface.

6.2 Winding Path Analysis

Our method is a semi-automatic method that requires the user to manually make adjustments. So, it is necessary to have principles to help the user evaluate the winding path quality.

We develope some tools to provide such interactive feedback based on geometry, generally via a color-coded map on the surface:

- **Angle of intersection:** When two fibers intersect, they provide strength in different directions. For example, if the fibers are only in one direction, the other direction cannot provide enough strength; the material could "tear" along the fiber path direction relatively easily. Generally, the more perpendicular the directions of two crossing fibers are, the stronger the resulting manufactured object. In Fig. 6.3, (b) is a good solution while (c) may not provide enough horizontal strength and (d) may not provide enough vertical strength. We define the angle of intersection's quality as the absolute cosine value of the intersection angle ($|\cos\theta|$ in Fig. 6.3-a).

- **Number of intersections:** The paths that our method generates create two families of curves with two different directions. So, a curve always intersects with curves in the other family and is roughly (or for symmetric surfaces, exactly) parallel to curves in the same family. Although angle of intersection is more important, generally speaking, the more intersections between families, the higher strength the result.

- **Path coverage:** Although we simplify the filament as a geometric curve, actual fibers in manufacturing are laid out in groups, and thus form "ribbons" with constant non-zero width. Coverage can be computed to determine the area of the mandrel covered by the filaments. We can display the variance of the coverage in different parts of the surface, highlighting based on the number of layers of filaments covering any one portion of the surface (in some direction). Approximately speaking, the closer a pair of parallel paths are, the greater the coverage provided by the solution. So we use the segment length between two parallel paths (d_1 and d_2 in Fig. 6.3-a) to represent the coverage of the winding paths.

  The coverage heavily depends on the local concavities, especially Gaussian curvature when the surface is locally convex. A larger Gaussian curvature usually "push away" paths harder, and the paths are sparser. Similar to the case shown in Fig. 4.1, if a curve locates near a locally convex area, the path is unstable, and will slide away under the tension. This will result a locally sparsity of winding path.

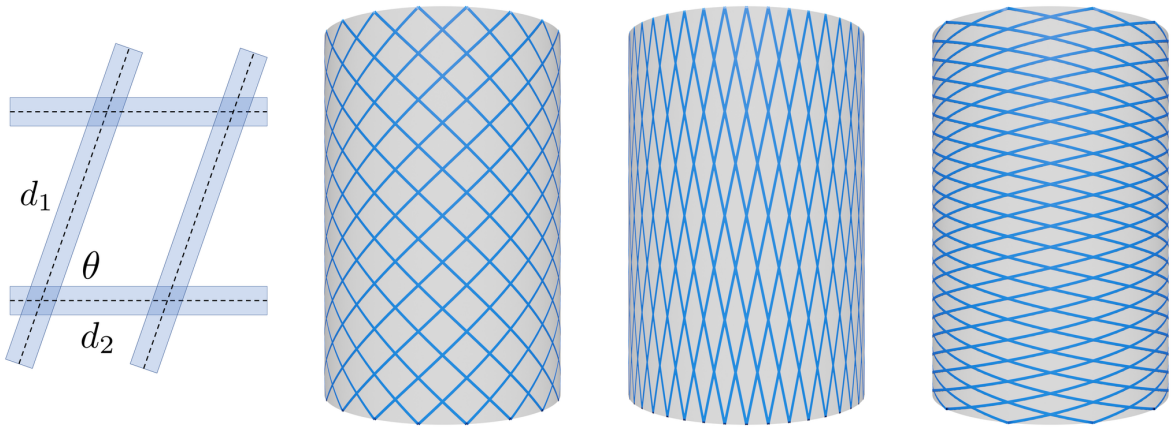


Figure 6.3: Winding path analysis. (a) is a sketch of winding path analysis. The winding paths generated by our method consist of two groups of parallel paths. The fiber width is l. The segment between two paths from another group has a length of d_1 and d_2, respectively. The angle of intersection is θ or $\pi - \theta$. (b) - (d) shows different angles of intersection.

Fig. 6.4 shows how our system visualizes the winding quality.

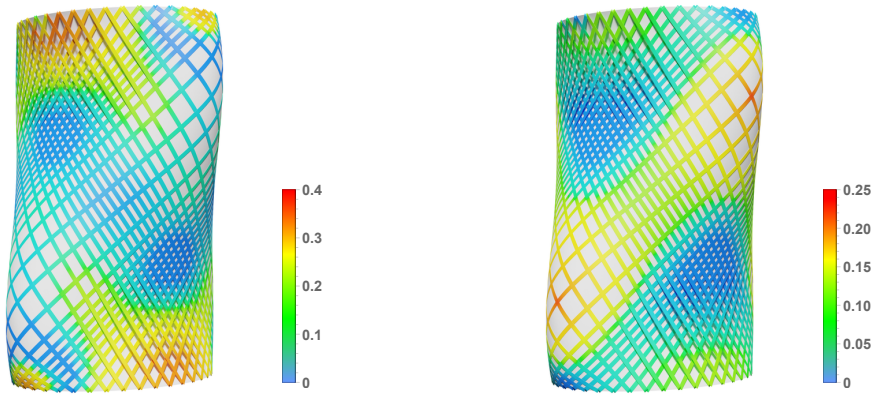
Users can judge the final results with the three principles described in this subsection and adjust the parameters such as the number of revolutions. But, a more extensive analysis might perform a full finite element simulation of the resulting structure, which is computationally expensive.

We implement an end-to-end filament winding system which supports surface editing, parameter control, path generation, path analysis, and winding process visualization. Fig. 6.5 is a demonstration of our system.

6.3 Non-Uniformly Distributed Paths

In Fig. 6.4-b, we notice that the coverage of winding paths can be significantly different all over the model. In the previous sections, we suppose that the end points of winding paths are evenly distributed. With non-uniformly distributed paths, the sparse area may be able to have a better coverage.

Suppose there are two adjacent paths in the same family of paths. As we described before, they don't cross each other. We use the distance measurement in section 6.2 to compute the distance



(a) (b)

Figure 6.4: Winding path evaluation and visualization. (a) shows the angle of intersection quality, and (b) shows the coverage quality.

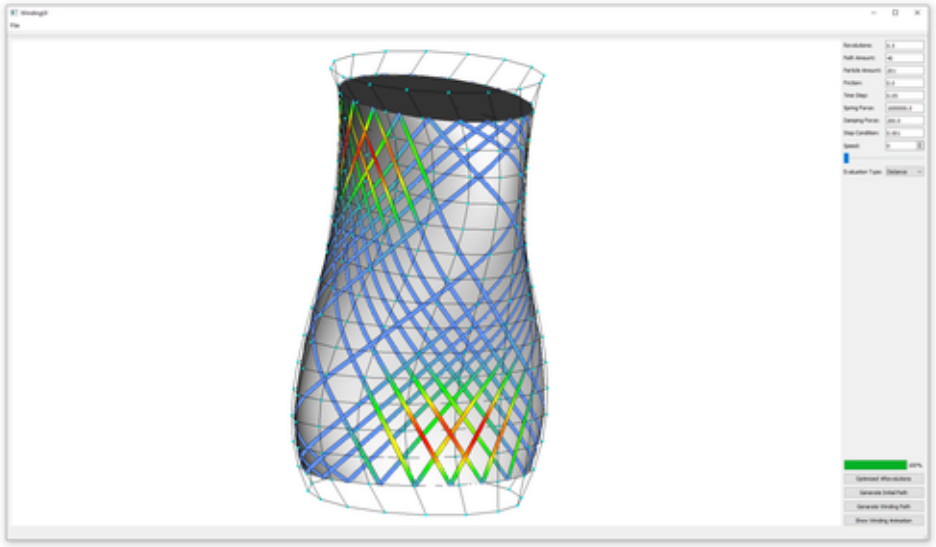


Figure 6.5: A demonstration of our end-to-end filament winding design system. Our design system provides winding path simulation and generation, winding animation, and simulation result analysis.

(a) Base Paths (40 Paths)

(b) Paths After Insertion (70 Paths)

(c) Uniformly Distributed Paths (70 Paths)

Figure 6.6: Uniformly distributed paths vs. non-uniformly distributed paths. (a) shows a sparser path configuration. Some gaps between paths are much larger than some others. After adaptively inserting new paths, the winding paths become (b). If we use the same number of paths but with uniform distribution, (c) is the result. It's obvious that the non-uniformly distributed paths (b) have smaller gaps than the uniformly distributed paths (c).

$dist$ between them. If $dist$ is larger than a user given threshold $dist_{\max}$, we insert $ceiling(\frac{dist}{dist_{\max}})$ new paths between the original two paths evenly, where $ceiling(x)$ means the minimum integer that is not smaller than x. Fig. 6.6 shows our result.

In Fig. 6.6, non-uniformly distributed paths form regular quadrilateral patterns with similar sizes. It is obvious that our method can also be used for quadrangulation. By connecting all path intersections, the paths become a quad-mesh of the original mandrel shape. Fig. 6.7 is a quadrangulation of Fig. 6.6-b using our method.

6.4 Automatic Surface Editing

Not all the surfaces have a feasible winding solution due to local concavities. The designer can edit the surface according to our analysis results, especially the distance from a path to the surface. However, it can be hard to adjust the surface and make it windable. So it's necessary to include an automatic shape editing tool in our end-to-end system.

Most parametric surface based shape modeling methods use control points for surface editing,

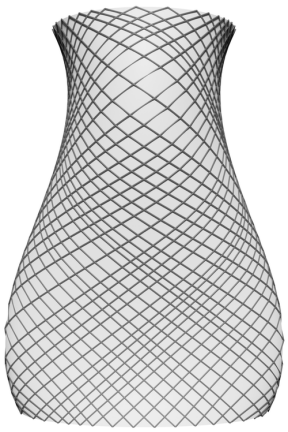


Figure 6.7: A quadrangulation example generated from non-uniformly distributed paths.

such as B-splines, NURBS, and some interpolation curves. Suppose the control points are $\mathbf{C} = \{C_i\}_{i=1}^{m}$. In this section, we explicitly denote the surface as

$$\vec{s}(\overline{P}; \mathbf{C}). \tag{6.4}$$

If the current path is lifted from the surface, we can adjust $\mathbf{C}$ to make the path on surface. Fig. 6.8 is the side view of the surface. The path is lifted due to the local concavities. P_i is on the lifted path, and $\vec{s}(\overline{P}_i; \mathbf{C})$ is a point on the shadow path. $\vec{n}_s(\overline{P}_i; \mathbf{C})$ is the surface normal at $\overline{P}_i$. Obviously, we have

$$(P_i - \vec{s}(\overline{P}_i; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_i; \mathbf{C}) > 0. \tag{6.5}$$

We move control points $\mathbf{C}$ to $\mathbf{C}^{new}$, then the point on the shadow path corresponding to P_i becomes $\vec{s}(\overline{P}_i; \mathbf{C}^{new})$. If $\vec{s}(\overline{P}_i; \mathbf{C}^{new})$ is above the lifted path, where P_i is, the surface is locally convex at $\overline{P}_i$. That is:

$$(\vec{s}(\overline{P}_i; \mathbf{C}^{new}) - \vec{s}(\overline{P}_i; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_i; \mathbf{C}) \geq (P_i - \vec{s}(\overline{P}_i; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_i; \mathbf{C}). \tag{6.6}$$

We will find $\mathbf{C}^{new}$ such that for all off-surface particles, Eq. 6.6 is true. However, the formulation

of $\vec{s}(\overline{P}_i; \mathbf{C}^{new})$ can be complicated and Eq. 6.6 is hard to solve directly. So we approximate Eq. 6.6 with a simpler form. Denote

$$\begin{aligned} \Delta\mathbf{C} &= \mathbf{C}^{new} - \mathbf{C}, \\ J_{\mathbf{C}}(\overline{P}; \mathbf{C}) &= \frac{\partial \vec{s}(\overline{P}; \mathbf{C})}{\partial \mathbf{C}}, \end{aligned} \tag{6.7}$$

where $J_{\mathbf{C}}(\overline{P}; \mathbf{C})$ is a 3 by $3m$ Jacobian matrix. m is the number of control points. Then

$$\vec{s}(\overline{P}_i; \mathbf{C}^{new}) - \vec{s}(\overline{P}_i; \mathbf{C}) \approx J_{\mathbf{C}}(\overline{P}_i; \mathbf{C})\Delta\mathbf{C}. \tag{6.8}$$

So Eq. 6.6 can be approximated by

$$\vec{n}_s(\overline{P}_i; \mathbf{C})^T J_{\mathbf{C}}(\overline{P}_i; \mathbf{C})\Delta\mathbf{C} \geq (P_i - \vec{s}(\overline{P}_i; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_i; \mathbf{C}). \tag{6.9}$$

By collecting all off-surface particles $P_{i_1}, P_{i_2}, \ldots, P_{i_k}$, we can build a system of linear inequalities:

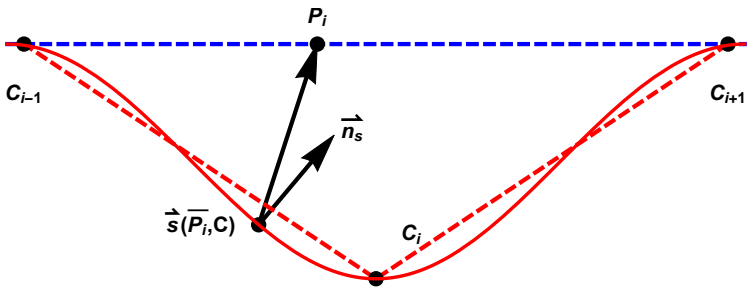
$$\begin{pmatrix} \vec{n}_s(\overline{P}_{i_1}; \mathbf{C})^T J_{\mathbf{C}}(\overline{P}_{i_1}; \mathbf{C}) \\ \vec{n}_s(\overline{P}_{i_2}; \mathbf{C})^T J_{\mathbf{C}}(\overline{P}_{i_2}; \mathbf{C}) \\ \vdots \\ \vec{n}_s(\overline{P}_{i_k}; \mathbf{C})^T J_{\mathbf{C}}(\overline{P}_{i_k}; \mathbf{C}) \end{pmatrix} \Delta\mathbf{C} \geq \begin{pmatrix} (P_{i_1} - \vec{s}(\overline{P}_{i_1}; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_{i_1}; \mathbf{C}) \\ (P_{i_2} - \vec{s}(\overline{P}_{i_2}; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_{i_2}; \mathbf{C}) \\ \vdots \\ (P_{i_k} - \vec{s}(\overline{P}_{i_k}; \mathbf{C})) \cdot \vec{n}_s(\overline{P}_{i_k}; \mathbf{C}) \end{pmatrix}, \tag{6.10}$$

and denote it as

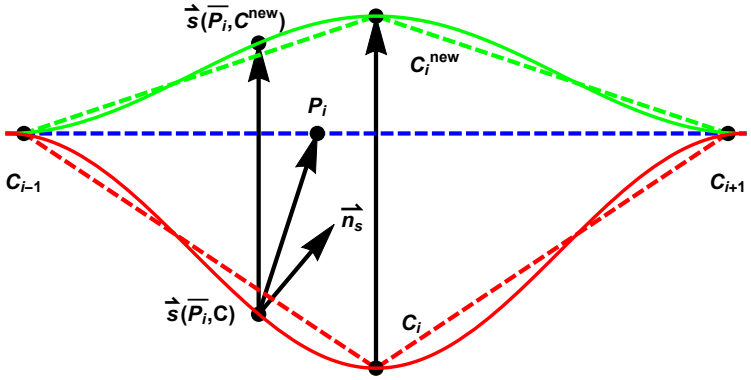
$$\mathbf{M}\Delta\mathbf{C} \geq \mathbf{B}. \tag{6.11}$$

$\mathbf{M}$ is a k by $3m$ matrix and $\mathbf{B}$ is a length k vector.

The editing should be as little as possible. So we can formulate the editing problem as a



(a)



(b)

Figure 6.8: Control point modification. In (a), C_{i-1}-C_i-C_{i+1} is a part of the control polygon. The red curve is a slice of the surface. Space above the red curve is the outside of the surface. P_i is an off-surface particle and it's corresponding shadow point is $\vec{s}(\overline{P}_i;\mathbf{C})$. Suppose we move C_i to C_i^{new} like (b), the control polygon becomes C_{i-1}-C_i^{new}-C_{i+1} and the green curve is a slice of the new surface. Now the shadow point $\vec{s}(\overline{P}_i;\mathbf{C}^{new})$ goes "above" the lifted path (the blue dash line), and $\vec{s}(\overline{P}_i;\mathbf{C}^{new})$ becomes an on-surface particle.

optimization problem:

$$\begin{aligned} \min_{\Delta\mathbf{C}} \quad & \Delta\mathbf{C}^T\Delta\mathbf{C} \\ s.t. \quad & \mathbf{M}\Delta\mathbf{C} \geq \mathbf{B}. \end{aligned} \tag{6.12}$$

This is a convex quadratic programing optimization and can be solved directly.

Because any particle can change its position in this optimization, the optimization cannot guarantee that the optimized curve is convex. So we repeat the optimization for several steps until the current path is totally convex.

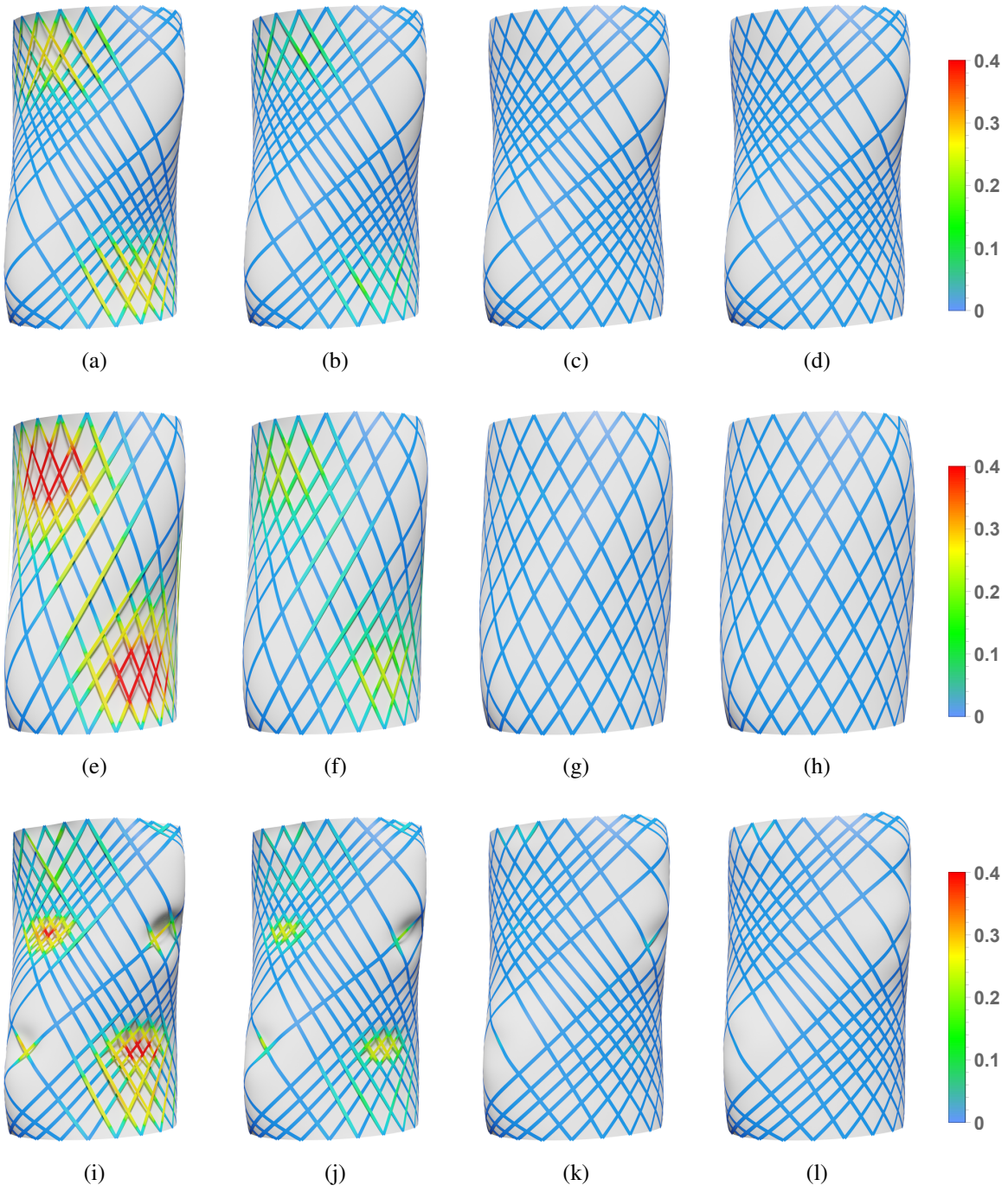


Figure 6.9: Automatically edited surfaces. The numbers of revolutions for (a)-(d) and (e)-(h) are 0.5 and 0.3, respectively. (i)-(l) is a different model. (a), (e) and (i) are the initial winding results. (b), (f) and (j) are the results after the first iteration of optimization. (c), (g) and (k) are results after 5 iterations. (d), (h) and (l) are results after 10 iterations.

7. RESULTS

In this section, we show that our method applies to more complex situations, and we compare our method with other methods.

7.1 Modeling Results

Besides static boundary conditions, our path generation method can also handle moving boundaries, which allow for representing a broader range of filament motions. Fig. 7.1 shows examples for a valley surface with a peak inside. The path's ends move along each side of the valley, and the fiber is caught by the peak. The first and the third rows of Fig. 7.1 show the results using our on-surface method without lifting; the second and the fourth rows show our off-surface algorithm with path lifting. Notice that the on-surface solver is always stuck by the peak, while the off-surface solver can jump over a smaller peak (the second row) and be stuck on a larger peak (the fourth row). It is obvious that the off-surface solver is more realistic.

B-spline surfaces are very general parametric surfaces and are widely used. Fig. 7.2 shows an example of applying our method to a B-spline surface. Fig. 7.2-a shows a B-spline surface with its control grid. We pick two points$P_1 = f(\overline{P}_1)$ and $P_n = f(\overline{P}_n)$ at each end of the surface and use them as the boundary. The initial path is given by mapping a straight line in the parametric space to the model space. That is, the curve's expression is $P(t) = f((1-t)\overline{P}_1 + t\overline{P}_n)$. After applying our method to it, we can generate a winding path in Fig. 7.2-b. Due to the concavity of the surface, there is some bridging. We use different colors to indicate the distance from the lifted curve to the surface. By repeating this step with different boundary settings, we can get winding paths in a regular patten, which is impossible for a forward solver.

Fig. 7.2-c also shows how our approach can be used to give feedback to a designer. If a concavity is found, designers can edit the B-spline surface according to the analysis. In this example, the designer moves the red control points in Fig. 7.2-a to the positions in Fig. 7.2-d. The subsequent path generation (Fig. 7.2-e) shows that there is no more bridging in that region, and that other paths

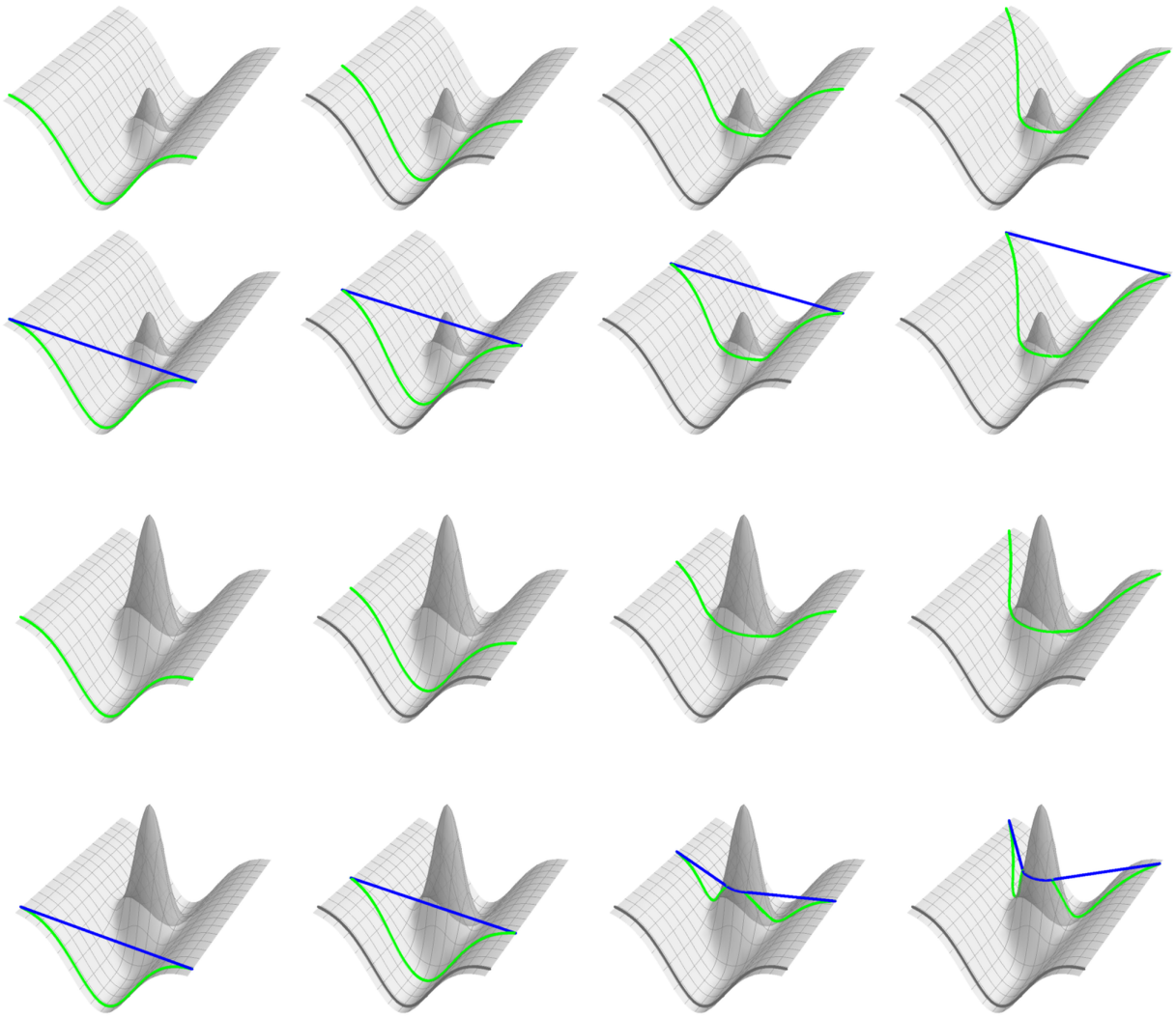


Figure 7.1: Examples with dynamic boundary conditions. The boundaries move at a constant speed. From left to right, each column represents a time step at 0.0s (initialization), 3.0s (300 steps), and 5.0s (500 steps). The first and second rows are the same as the third and fourth rows, but with a smaller peak. The first and third rows use the on-surface algorithm. The gray curves are the initializations, and the green curves are simulation results. The curve will be blocked by the peaks. Most current iterative methods for geodesic solving will be in the same situation. The second and fourth rows use our off-surface method. The gray curves are the initializations, the green curves are shadow paths, and the blue curves are the results. With the off-surface algorithm, a curve can fly over smaller peaks while wrapping around higher peaks.

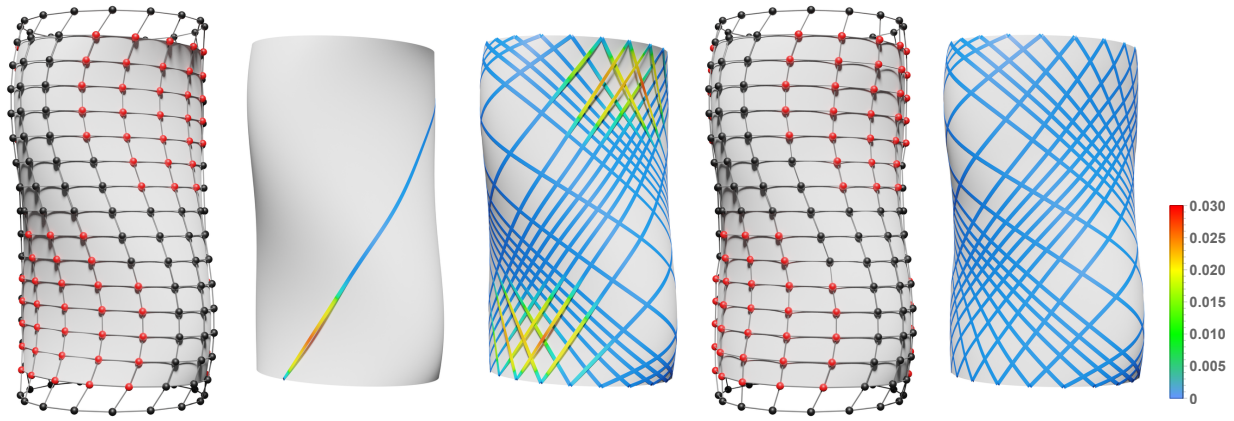


Figure 7.2: A B-spline surface example. (a) is the surface and its control grid. (b) is a single path result computed by our algorithm. The color on it shows the distance from the curve to the surface. By repeating (b) several times with different initial paths, a regular grid (c) will be generated. There are still some off-surface parts (marked as red). A designer can then move all red control points in (a) to get a modified model (d). The simulation results in (e) show the bridging is eliminated.

are changed only minimally.

We also test our algorithm's computation time. We implemented the algorithm in C++ with OpenMP and tested it on 6 threads of a PC with a Core i7 7700K processor. To generate Fig. 7.2-b, we ran 1000 iterations (time steps) to obtain a single path, and the time spent is listed in Table 7.1.

From the table, we notice that the most significant parts are building and solving the linear system. To reduce the computation time, we can reuse the same (factored) stiffness matrix for several consecutive steps, as the stiffness matrix changes little, allowing us to get rid of the two biggest bottlenecks. In section 7.2, we show that by reusing the stiffness matrix the method can converge faster while maintaining an accurate result. The remaining time consuming part is the lifting and landing algorithm. In Algo. 2, all off-surface particles need a nearest point calculation in each step. If our method is used for finding an on-surface geodesic, or the surface is guaranteed to be convex, it is not necessary to use the lifting algorithm, and we can save an additional 25% of the time.

To verify the accuracy of our winding calculation, we also 3D printed models of some shapes

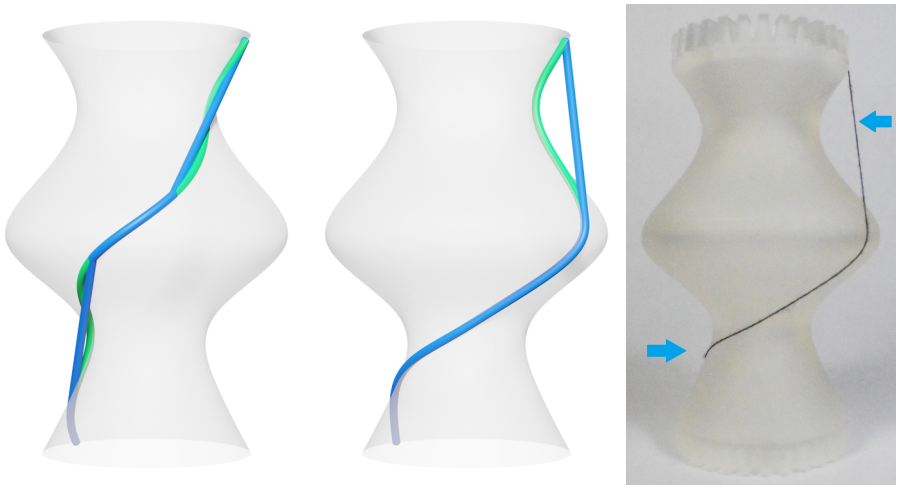
	Timing (ms)	Percentage
Building the Linear System	233.22	38.67%
Solving the Linear System	204.35	33.88%
Lifting and Landing	149.65	24.81%
Other Operations	15.86	2.63%
Total	603.08	100%

Table 7.1: Timing results of 1000 time steps. The table shows the time spent in each part of our method. About 70% of the time is spent on building and solving the motion equation, which is a sparse linear system. The timing result for lifting and landing includes Algo. 1 and Algo. 2. All other parts are not significant and are counted in Other Operations.

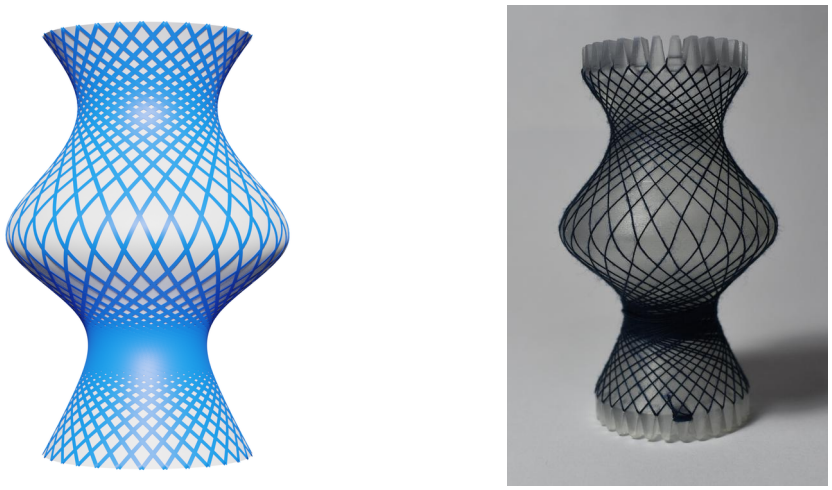
and wound thread to compare the real result to the prediction. Fig. 7.3 shows a vase with significant concavities, which our method handles effectively. Fig. 7.4 shows more 3D-printed examples. Our method can be applied to a wider range of practical shapes than which has traditionally been done in filament winding, such as the glider wing or leg cast shown in Fig. 7.4. We also show that our method can handle much more complex shapes in Fig. 7.5.

7.2 Comparisons

There are plenty of methods to simulate strings or rods. In filament winding modeling, the physical simulation of a winding path is significantly different from most of the string or rod simulation problems. The physical simulation of a winding fiber is almost irrelevant to twisting or bending, which can make the problem much more complicated. Furthermore, the fiber is mostly constrained on the surface. Thus, many rod simulation methods are significantly more complex and slower than necessary. So, we compare our method only to two commonly used methods for constraining a 3D strand onto a surface: the penalty force method (PF) and the Karush–Kuhn–Tucker method (KKT). The PF method uses stiff penalties to push particles away from the surface, and the KKT method approximates the contact constraint as bilateral constraints, which can be turned on/off by checking the sign of the Lagrange multipliers. For both of these comparison methods (PF and KKT), we ignore twisting and bending energies, just like in our method. We do not consider linear complementarity or quadratic programming approaches, since they are significantly more expensive.

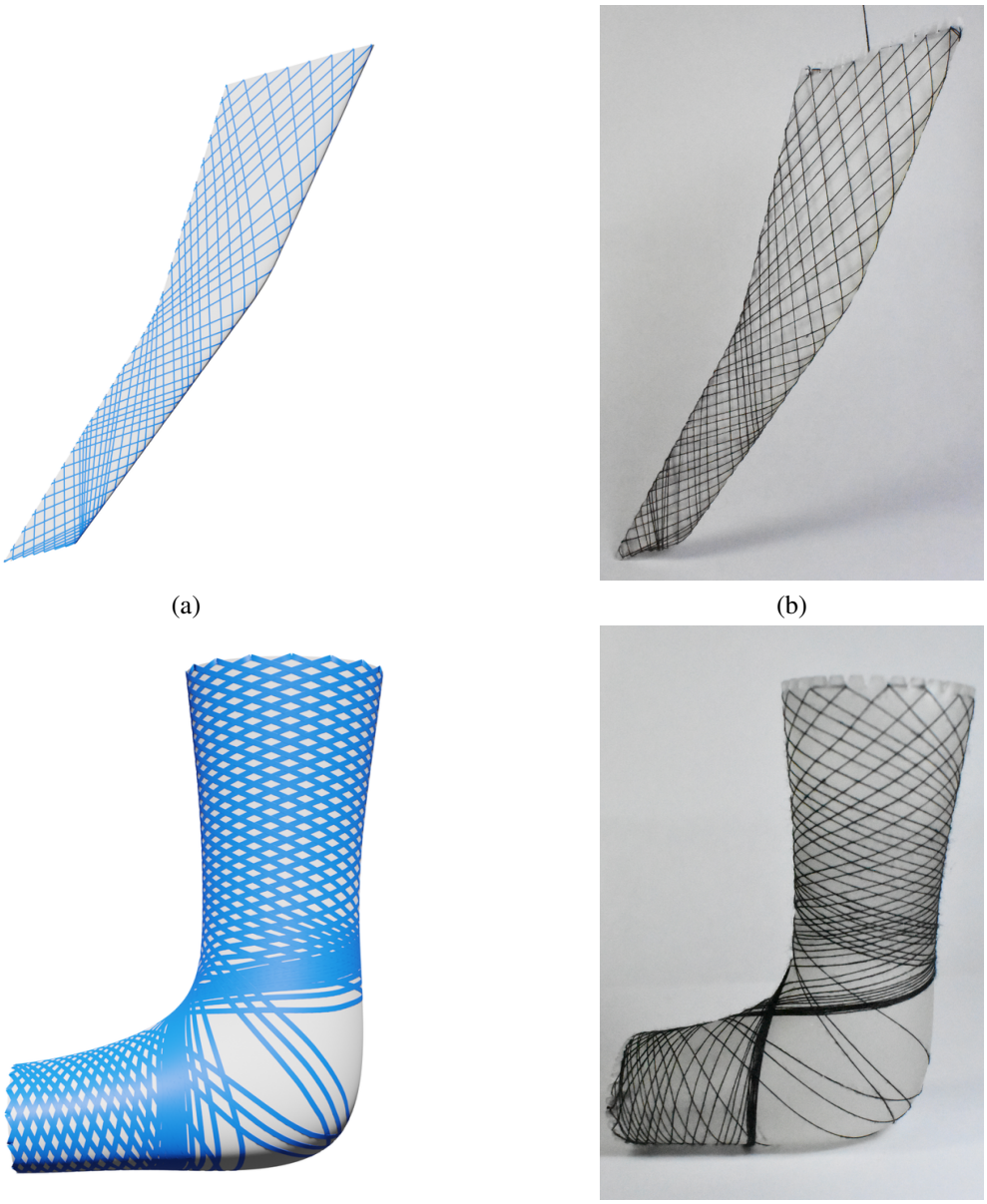


(a) (b) (c)



(d) (e)

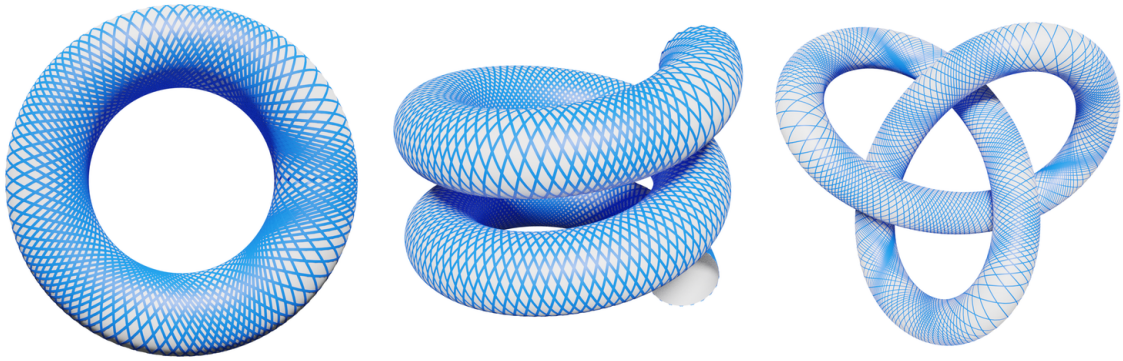
Figure 7.3: A vase example with concavities. The green curve in (a) is an initialization. The blue path shows which points are lifted according to concavity. (b) is the simulation result with the initialization in (a). The green curve is the shadow path and the blue path is the final result. Some of the particles landed on the surface while some others left the surface. (c) shows the winding path on a real 3D-printed part. The path matches our result (b). With better initialization, the path can always stay on the surface as seen in (d). (e) is a 3D printed result of (d).



(a) (b)

(c) (d)

Figure 7.4: Examples of applying our method to real models. (a) and (b) show a glider wing model. (c) and (d) show a leg cast model. The models were wound by hand and the friction may be different. So the winding results may be slightly different from the simulation results. If a winding machine is used, the winding results can be closer to the simulation results.



(a) (b) (c)

Figure 7.5: More complex examples using our method.

The first method is the penalty force (PF) method. In this method, we simulate the fiber in 3D. When particles are far away from the surface, there is only the spring force. When the distance between a particle and the surface is smaller than a threshold ε_{PF}, a penalty force

$$F(d) = \begin{cases} 0, & d > \varepsilon_{PF} \\ k_{PF}\frac{\varepsilon_{PF}-d}{\varepsilon_{PF}}, & \varepsilon_{PF} \geq d > 0 \\ k_{PF}, & 0 \geq d \end{cases} \tag{7.1}$$

will be applied to the particle. d is the distance from the particle to the surface. d is negative when the particle is inside the surface. The direction of the penalty force is defined by the direction from the surface to the particle.

The second method is the Karush–Kuhn–Tucker(KKT) conditions method. If the system has no constraints, we can formulate the simulation into a linearly implicit scheme

$$(M - h^2K)v = Mv_0 + hf. \tag{7.2}$$

In the KKT method, when a particle touches the surface, we apply a surface constraint to the

particle:

$$n \cdot v = 0, \tag{7.3}$$

where n is the surface normal at the particle, and v is the particle's velocity. We can collect all the constraints into a global matrix G, such that

$$Gv = 0. \tag{7.4}$$

Solving Eq. 7.2 under the constraint Eq. 7.4 is equivalent to an optimization problem:

$$\min_{v} \quad \frac{1}{2} v^T \tilde{M} V - v^T \tilde{f}, \tag{7.5}$$

$$s.t. \quad Gv = 0. \tag{7.6}$$

The solution of Eq. 7.6 can be computed from

$$\begin{pmatrix} \tilde{M} & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} v \\ \lambda \end{pmatrix} = \begin{pmatrix} \tilde{f} \\ 0 \end{pmatrix}. \tag{7.7}$$

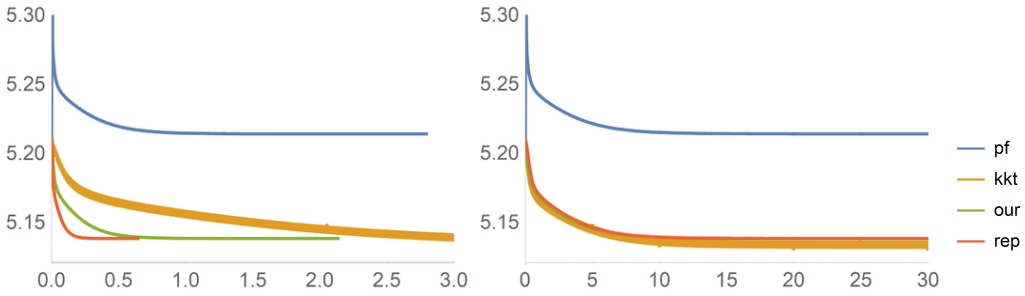
Then λ can be computed from

$$G\tilde{M}G^T\lambda = G\tilde{M}\tilde{f}, \tag{7.8}$$

and v can be computed from

$$\tilde{M}v = \tilde{f} - G^T\lambda. \tag{7.9}$$

$\tilde{M}$ can be pre-factorized for faster computation. λ indicates the constraint force. When $\lambda > 0$, the constraint force is pointing inside of the surface. In other words, the particle is leaving the surface.



(a) Wall-clock Time (b) Simulation Time

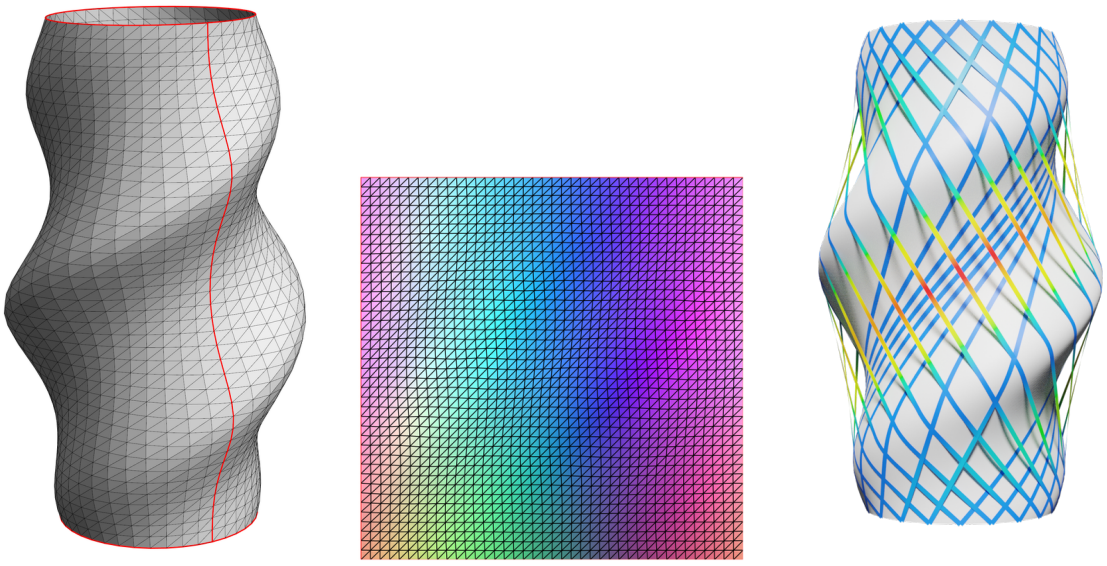
Figure 7.6: Timing and convergence comparison. In (a), the X axis is wall-clock time (in seconds), and in (b), the X axis is simulation time (in seconds). The Y axes in both (a) and (b) are the total length of the simulated path (in cm). The figure shows the convergence of length using different methods. According to (a), our method is faster than KKT, and more accurate than PF. According to (b) our method has similar simulation behavior as KKT.

In this case, there should be a constraint. So all positive terms will be reset to 0. Otherwise, the path will "stick" to the surface.

We compare the three methods as well as our method with reusing stiffness matrix (rep) in Fig. 7.6. In the figures, we use the total length of the path to evaluate the convergence. According to the figures, our method and the KKT method produce almost identical results. The penalty force approach always has some errors. Because the penalty force pushes the path away from the surface when the path is very close (ε_{PF}) to the surface, the error is highly related to ε_{PF}. When ε_{PF} is large, the final result is far away from the ground-truth. When ε_{PF} is small, the method is unstable. The KKT method is more robust, but converges slower in wall-clock time. Furthermore, similar to the PF method, the KKT method requires parameter tuning (stabilization parameter [2]), though nowhere near as much as the PF method. Our method can get an accurate result while having the fastest computation speed, and does not require any parameter tuning for constraint handling. By reusing the stiffness matrix, our method is even faster while the result is still accurate.

7.3 Freeform Mesh

Although parametric surfaces are flexible enough for shape designing, freeform meshes are widely used. So we extend our method to freeform meshes. Parameterization is a way to convert a



(a) Freeform Mesh (b) Parameterization (c) Winding Result

Figure 7.7: A freeform mesh example. Given a freeform mesh (a), we can parameterize it as (b). (b) shows the parameterized mesh in the parametric space. A pixel's RGB value represents the XYZ coordinate of a surface point. Using our method, we can generate paths as in (c).

freeform mesh to a parametric surface.

One of our method's basic assumption is that the surface should be isotopic to a cylinder. For a closed mesh, first we choose two ends of the shape and map them to a cylinder's ends. Then the surface is cut through one end to the other end. The mesh can be parameterized as a rectangle in the parametric space [10]. Without loss of generality, suppose the mesh is mapped to $[0, 1] \times [0, 1]$. Then the mesh can be mapped to a cylinder by gluing two edges of the parameter space rectangle.

With a parameterized surface, we can compute the winding path using our method. Fig. 7.7 is an example of a freeform mesh.

8. CONCLUSION

We have presented a method for computing winding paths in the presence of concavities and accounting for friction. This enables us to simulate path layouts that correspond to real behavior in filament-wound parts. More specifically, we have introduced a physically-based model ideally suited for this application that can incorporate dynamic friction (not just static analysis) into the process. We have also shown how we can augment this simulator to model filaments that bridge across concavities, without having to implement costly collision detection and resolution. Our resulting end-to-end system demonstrates that we can handle dynamic behavior of filament winding across a variety of surfaces with friction and concavities, that the simulations we achieve are close to the real-world winding paths, and that we can provide guidance to a designer to generate windable parts.

The thesis of the dissertation states that:

The filament winding problem can be solved with physically based fiber simulation in the 2D parametric space. With the method we propose, the winding solution can be both accurate and fast to generate.

We prove the thesis by the following aspects:

- Existing filament winding path generation methods focus on simple shapes such as cylinders and elbows. Our method significantly extended the range of mandrel shapes, which is not explored in previous works. With our work, filament winding can be used in more applications with complex shapes. The shapes can be parametric surfaces or free-form meshes with parameterization, which include most of the commonly used shapes.

- This work also makes the winding path generation fast enough for designers to see the simulation result in seconds. This lets the designers work more efficiently. The results are also reliable and very similar to the real-world winding result, as our method also takes friction and concavity into consideration.

- Our analysis framework is another main contribution of this work. With our analysis framework, designers and users can evaluate winding results more intuitively, and then edit the winding paths or shapes by themselves or our automated methods according to the analysis result. The framework also provides an evaluation method for other filament winding methods beyond this work.

A motivating goal of our work is to support comprehensive design for a much wider range of filament-wound parts than is now produced in practice. Although just one piece, our path generation is the most critical aspect of achieving such a system. There remain several related avenues for further work; among these directions are:

- Our basic end-to-end system could be extended to address a higher level of usability challenges, such as incorporating structural analysis or performing more generalized shape optimization on the model.

- Our current winding paths do not take into consideration some of the fully global issues, such as accounting for a particular physical winding machine geometry to ensure the head positioning orientation. Such considerations might put additional constraints on the path generation problem, and also present a different separate challenge of possible machine design and generating machine instructions.

- Our approach as described uses a parametric surface. Although we show our method can be extended to general freeform meshes, there are also non-trivial challenges faced in such cases (e.g., virtually cutting a mesh for multiple revolutions of winding, or defining fixation points) that would need to be addressed before it could be considered solved.

REFERENCES

[1] F. Abdel-Hady. Filament winding of revolution structures. *Journal of Reinforced Plastics and Composites*, 24(8):855–868, 2005.

[2] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computer Methods in Applied Mechanics and Engineering*, 1(1):1–16, 1972.

[3] M. Bergou, M. Wardetzky, S. Robinson, B. Audoly, and E. Grinspun. Discrete elastic rods. In *ACM SIGGRAPH 2008 Papers*, pages 1–12. Association for Computing Machinery, 2008.

[4] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 594–603. Association for Computing Machinery, 2002.

[5] J. Brown, J.-C. Latombe, and K. Montgomery. Real-time knot-tying simulation. *The Visual Computer*, 20(2-3):165–179, 2004.

[6] J. Chen and Y. Han. Shortest paths on a polyhedron. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, page 360–369, New York, NY, USA, 1990. Association for Computing Machinery.

[7] V. Costalonga Martins, S. Cutajar, C. van der Hoven, P. Baszyński, and H. Dahy. Flexflax stool: Validation of moldless fabrication of complex spatial forms of natural fiber-reinforced polymer (NFRP) structures through an integrative approach of tailored fiber placement and coreless filament winding techniques. *Applied Sciences*, 10(9):3278, 2020.

[8] K. Crane, C. Weischedel, and M. Wardetzky. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics*, 32(5):1–11, 2013.

[9] J. De Carvalho, M. Lossie, D. Vandepitte, and H. Van Brussel. Optimization of filament-wound parts based on non-geodesic winding. *Composites Manufacturing*, 6(2):79–84, 1995.

[10] M. S. Floater. Parametrization and smooth approximation of surface triangulations. *Computer Aided Geometric Design*, 14(3):231–250, 1997.

[11] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. *Advances in Multiresolution for Geometric Modelling*, pages 157–186, 2005.

[12] J. Fu, J. Yun, and Y. Jung. Filament winding path generation based on the inverse process of stability analysis for non-axisymmetric mandrels. *Journal of Composite Materials*, 51(21):2989–3002, 2017.

[13] J. Fu, J. Yun, Y. Jung, and D. Lee. Generation of filament-winding paths for complex axisymmetric shapes based on the principal stress field. *Composite Structures*, 161:330–339, 2017.

[14] M. Hofer and H. Pottmann. Energy-minimizing splines in manifolds. In *ACM SIGGRAPH 2004 Papers*, pages 284–293. Association for Computing Machinery, 2004.

[15] T. Kanai and H. Suzuki. Approximate shortest path on a polyhedral surface based on selective refinement of the discrete graph and its applications. In *Proceedings Geometric Modeling and Processing 2000. Theory and Applications*, pages 241–250. IEEE, 2000.

[16] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proceedings of the National Academy of Sciences*, 95(15):8431–8435, 1998.

[17] S. Koussios, O. K. Bergsma, and G. Mitchell. Non-geodesic filament winding on generic shells of revolution. *Proceedings of the Institution of Mechanical Engineers, Part L: Journal of Materials: Design and Applications*, 219(1):25–35, 2005.

[18] H.-s. Li and Y.-d. Liang. Computer aided filament winding for elbows. *Journal of Software*, 13(4):518–525, 2002.

[19] H.-s. Li, Y.-d. Liang, and H. Bao. Cam system for filament winding on elbows. *Journal of Materials Processing Technology*, 161(3):491–496, 2005.

[20] C.-C. Liang, H.-W. Chen, and C.-H. Wang. Optimum design of dome contour for filament-wound composite pressure vessels based on a shape factor. *Composite structures*, 58(4):469–482, 2002.

[21] B. Liu, S. Chen, S.-Q. Xin, Y. He, Z. Liu, and J. Zhao. An optimization-driven approach for computing geodesic paths on triangle meshes. *Computer-Aided Design*, 90:105–112, 2017.

[22] J. Marketos. Optimum toroidal pressure vessel filament wound along geodesic lines. *AIAA Journal*, 1(8):1942–1945, 1963.

[23] D. Martínez, L. Velho, and P. C. Carvalho. Computing geodesics on triangular meshes. *Computers & Graphics*, 29(5):667–675, 2005.

[24] S. T. Peters. *Composite Filament Winding*. ASM International, 2011.

[25] S. Pillwein, K. Leimer, M. Birsak, and P. Musialski. On elastic geodesic grids and their planar to spatial deployment. *ACM Transactions on Graphics*, 39(4):1–12, 2020.

[26] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *Discrete Differential Geometry: An Applied Introduction*, pages 30–38. ACM SIGGRAPH Course Notes, 2006.

[27] Y. Qin, X. Han, H. Yu, Y. Yu, and J. Zhang. Fast and exact discrete geodesic computation based on triangle-oriented wavefront propagation. *ACM Transactions on Graphics*, 35(4):1–13, 2016.

[28] M. Rabinovich, T. Hoffmann, and O. Sorkine-Hornung. Discrete geodesic nets for modeling developable surfaces. *ACM Transactions on Graphics*, 37(2):1–17, 2018.

[29] M. Rabinovich, T. Hoffmann, and O. Sorkine-Hornung. The shape space of discrete orthogonal geodesic nets. *ACM Transactions on Graphics*, 37(6):1–17, 2018.

[30] M. Raffaelli, J. Bohr, and S. Markvorsen. Cartan ribbonization and a topological inspection. *Proceedings of the Royal Society A*, 474(2220):20170389, 2018.

[31] D. V. Rosato and C. S. Grove. *Filament Winding: Its Development, Manufacture, Applications, and Design*. Interscience Publishers, 1964.

[32] J. Scholliers. *Robotic Filament Winding of Asymmetric Composite Parts*. Ph.D. thesis, KU Leuven, 1992.

[33] S. Seereeram and J.-Y. Wen. An all-geodesic algorithm for filament winding of a T-shaped form. *IEEE Transactions on Industrial Electronics*, 38(6):484–490, 1991.

[34] A. Selle, M. Lentine, and R. Fedkiw. A mass spring model for hair simulation. In *ACM SIGGRAPH 2008 Papers*, pages 1–11. Association for Computing Machinery, 2008.

[35] A. Sheffer, E. Praun, and K. Rose. Mesh parameterization methods and their applications. *Foundations and Trends in Computer Graphics and Vision*, 2(2):105–171, 2006.

[36] L. Sorrentino, E. Anamateros, C. Bellini, L. Carrino, G. Corcione, A. Leone, and G. Paris. Robotic filament winding: An innovative technology to manufacture complex shape structural parts. *Composite Structures*, 220:699–707, 2019.

[37] S. Sueda, G. L. Jones, D. I. W. Levin, and D. K. Pai. Large-scale dynamic simulation of highly constrained strands. In *ACM SIGGRAPH 2011 Papers*, pages 1–10. Association for Computing Machinery, 2011.

[38] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Transactions on Graphics*, 24(3):553–560, 2005.

[39] E. Vargas Rojas, D. Chapelle, D. Perreux, B. Delobelle, and F. Thiebaud. Unified approach of filament winding applied to complex shape mandrels. *Composite Structures*, 116:805–813, 2014.

[40] J. Vekhter, J. Zhuo, L. F. G. Fandino, Q. Huang, and E. Vouga. Weaving geodesic foliations. *ACM Transactions on Graphics*, 38(4):1–22, 2019.

[41] S.-Q. Xin and G.-J. Wang. Efficiently determining a locally exact shortest path on polyhedral surfaces. *Computer-Aided Design*, 39(12):1081–1090, 2007.

[42] C. Xu, T. Y. Wang, Y.-J. Liu, L. Liu, and Y. He. Fast wavefront propagation (FWP) for computing exact geodesic distances on meshes. *IEEE Transactions on Visualization and Computer Graphics*, 21(7):822–834, 2015.

[43] B. Zhang, H. Xu, L. Zu, D. Li, B. Zi, and B. Zhang. Design of filament-wound composite elbows based on non-geodesic trajectories. *Composite Structures*, 189:635–640, 2018.

[44] L. Zu, S. Koussios, and A. Beukers. Design of filament–wound domes based on continuum theory and non-geodesic roving trajectories. *Composites Part A: Applied Science and Manufacturing*, 41(9):1312–1320, 2010.

[45] L. Zu, S. Koussios, and A. Beukers. Design of filament-wound circular toroidal hydrogen storage vessels based on non-geodesic fiber trajectories. *International Journal of Hydrogen Energy*, 35(2):660–670, 2010.