Modeling Natural Geometries Using Repulsive Shape Optimization

ME 6104 - Computer-Aided Design
Final Report
Professor: Dr. Yan Wang

April 29, 2022

Group Members:
Ryan Grajewski
Paul Babou

1 Introduction

1.1 Project Background and Motivation

Shape and topology optimization applications are frequently used when modeling complex 3D geometries in order to efficiently generate or improve existing 3D CAD models. These ordinary optimization schemes attempt to solve what is known as the optimal control problem, which involves finding a geometry, shape, or topology such that a certain desired parameter or cost functional is either maximized or minimized, while also satisfying other defined constraints[1]. It is common for a 3D model to be optimized for parameters such as weight of the part, material cost, surface area, topology, stress concentrations, manufacturability, and more. The reason optimization techniques have become a popular area of study is because of their potential impact on industry where time and material cost constraints make it necessary to optimize parts from the onset of the design phase, rather than iterating through a trial-and-error process.

In the context of most real-world design problems, an optimization study for a part must perform two key functions: it must maximize or minimize the desired parameters within the constraints, and it must also result in realistic geometries that can exist in the real world. While many programs are available for sufficiently performing the first function, the second function is often overlooked. The resulting optimized part can often contain impossible geometries like curve self-intersections or surface collisions. These geometries have the potential to generate invalid or non-orientable objects which do not follow the dimensional constraints we all live by. A good example of this is the Klein Bottle [Figure 1] – while it may look generic, the self-intersection makes the bottle a one-sided surface which is a phenomenon that can only exist in a four dimensional setting.

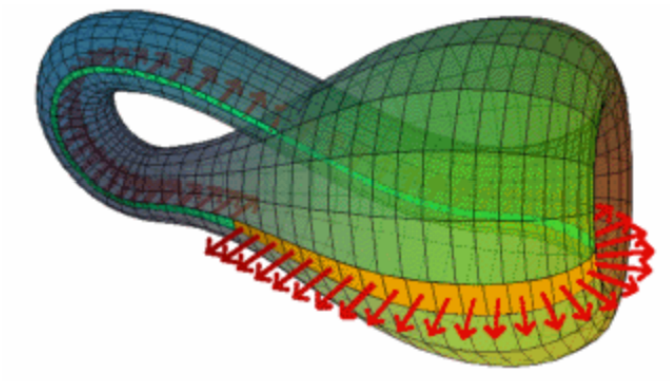


Figure 1: Klein Bottle – a non-orientable, two-dimensional closed manifold is an impossible topology because the self-intersection makes it a one-sided surface [2]

Repulsive optimization methods attempt to solve these challenges by inherently preventing curves or surfaces from intersecting or “crossing over” in space. This is done by employing a secondary cost functional, known as a repulsive energy, that translates or repels points that are close in space (high energy) away from each other while simultaneously solving the given optimal control problem. Repulsive optimization methods are therefore valuable tools for disciplines like mathematical visualization, image processing, rendering, animation, and computational design. One such application that is of particular interest, and is the motivation behind this project, is computer modeling of complex geometries that are found in nature. The laws of nature dictate that organisms and membranes must grow or form in such a way so that they avoid passing through themselves. Thus, a unique capability of repulsive optimization is the ability to model things such as neural networks, blood veins, muscle tissues, etc.

1.2 Project Objective

The driving objective for this project is to demonstrate how repulsive optimization theory can be used to accurately model a simplified bicep muscle tissue strand [Figure 2]. To narrow the scope of this endeavor, however, the immediate objective of this modeling project – and what this report principally focuses on – is developing a python optimization algorithm which can be used to eliminate self-intersections for any input closed curve. The designed algorithm should take in input control coordinates for a closed spline curve, and output a visualization of the input curve as well as the final optimized curve. Such an algorithm consists of two key functionalities: it (1) implements a repulsive energy function, calculating the repulsive energy values for all points along the curve, and (2) uses an optimization method to minimize that energy along the curve by iteratively translating the curve towards orientations of lower energy. While several alternatives exist, the repulsive energy function that is utilized for this algorithm is the repulsive Tangent-Point Energy. Similarly, there are multitudes of optimization schemes available for implementation, but for this project only Gradient Descent was considered because it is easy to implement and the focus of the project is the behavior of the repulsive energy.

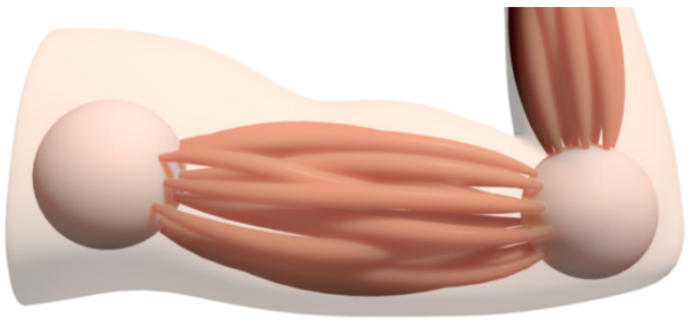


Figure 2: Theoretical results of future work – bicep muscle strands modeled as non-intersecting splines between two static control points[3]

1.3 Existing Approaches for Repulsive Energy Functions

Several repulsive energy functions have been documented that can be used for this curve optimization algorithm. In general, these functions should calculate an energy value for all points along a curve, and energies should approach infinity as the distance in space to another point along the curve approaches zero.

The Coulomb-Electrostatic Energy is one of the more rudimentary forms of energy functions as it is an adaptation of Coulomb's Law, which quantifies the force between two stationary electrically charged particles. In the context of curves, Coulomb-Electrostatic Energy can be thought of as the electrical repulsive force between two points on a curve if that curve were given a uniformly distributed electrical charge[4]. A good visual for this energy function is shown below in Figure 3.

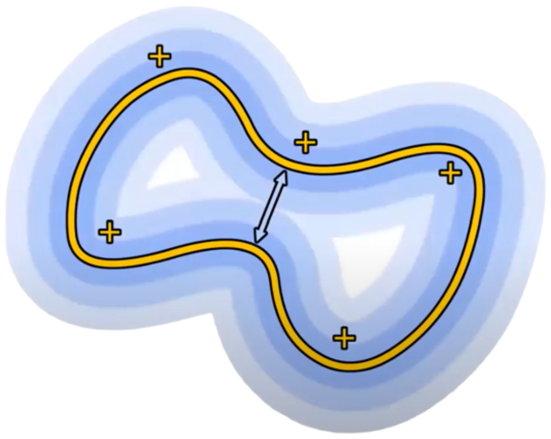


Figure 3: Visualization of Coulomb-Electrostatic Energy. A curve is given a uniform electric charge, forcing points to repel each other due to their like charges.

With Υ denoting a closed-curve, the equation for the Coulomb-Electrostatic Energy is:

$$\epsilon_{coulomb} = \iint \frac{1}{|\gamma(x_1) - \gamma(x_2)|^{\alpha}}$$

which translates as the inverse of the distance between two points along a curve raised to a power α. The exponent α dictates the rate at which the energy loses its strength as distance increases. The drawback for using this energy function is that it also considers adjacent points along a curve, applying massive energy values to those points because they are considered closest[4]. Also, for $\alpha < 2$ the energy cannot prevent intersections. Optimizing this energy would not be advantageous for this algorithm because it would not specifically target self-intersections, and would attempt to repel points that are next to each other along a curve as well.

The second energy function considered for this algorithm is the Möbius Energy function which was first defined by O'Harra in 1989 for use in knot theory. This energy function expands upon the Coulomb-Electrostatic Energy function by introducing a correctional component that removes the influence of nearby and adjacent points along the curve[5]. The equation for Möbius Energy is:

$$\epsilon_{Möbius} = \iint \frac{1}{|\gamma(x_1) - \gamma(x_2)|^2} - \frac{1}{[D(x_1, x_2)]^2}$$

where the correctional component is D(x1,x2) which is defined as the geodesic distance along the curve[6]. This energy function is more robust than the Coulomb-Electrostatic Energy because it ensures that adjacent points along a curve will not repel each other. However, a key aspect of this energy which happens to be a drawback for this algorithm is that it is invariant to Möbius transformations. Möbius Invariance means that points will remain the same after a Möbius transformation, and for our purpose that means that the energy function purposefully ignores sections of a curve where points are very close in space but only slightly distant along the curve – picture a tightly wound rope[7]. While this drawback could easily be controlled by avoiding certain input curves for the algorithm, it presents an inherent and upfront limitation.

The repulsive Tangent-Point Energy function was selected for this algorithm because it mitigates the concerns found with both the Coulomb-Electrostatic Energy and Möbius Energy functions. The equation for Tangent-Point energy is:

$$\epsilon_{Tangent-Point} = \iint \frac{1}{R_t(x,y)^{\alpha}}$$

where R(x,y) is the radius of the smallest sphere passing through a point Y that is also tangent to a point X along the same curve[6]. A good visualization of this sphere is shown in Figure 4. By definition, a larger radius means a lower energy, and vice versa. This energy function inherently minimizes contributions from adjacent points along a curve because the sphere connecting the two would need to be massive, and therefore produce an inconsequential energy. Furthermore, Tangent-Point energy is not Möbius Invariant.

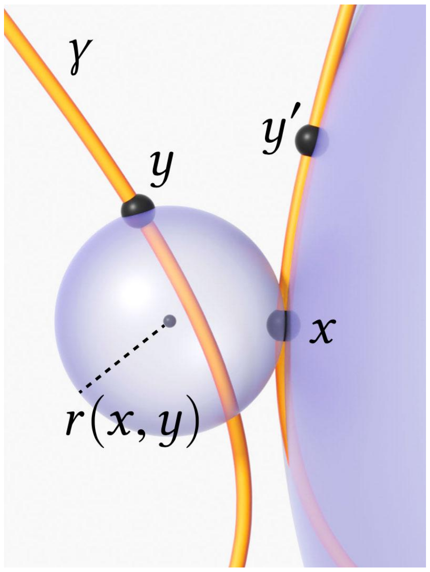


Figure 4: Visualization of Tangent-Point Energy[3]

2 Approach and Methodology

The following section provides an in-depth, step-by-step of the implementation approach for repulsive energy function and optimization method. In general, the approach for developing our repulsive optimization algorithm can be separated into two steps: Tangent-Point Energy, and Gradient Descent. The methods and equations described in this section are all coded in Python using similar logic.

The closed spline curve shown in Figure 5 is used as the functionality test case as it is easy to visualize where the points of highest energy are located, and where they should move during optimization. This curve is initialized by inputting 11 point coordinates into the Parametric Spline function within the python visualization toolkit (VTK), which interpolates a smooth discrete curve between each coordinate. To alleviate computational time while still maintaining an accurate tangent line for the Tangent-Point Energy sphere, the parametric spline function only generates 200 discrete points to define the curve.

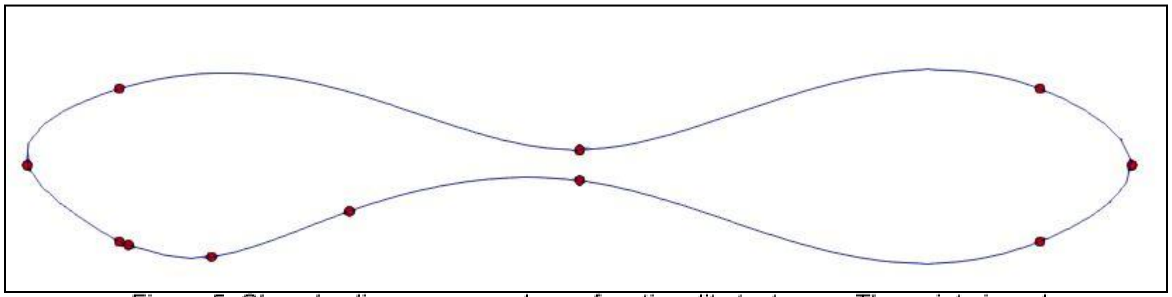


Figure 5: Closed spline curve used as a functionality test case. The points in red are the spline input coordinates.

2.1 Implementation of Tangent-Point Energy

2.1.1 Development of Radius Equation

The first and most important step in implementing the Tangent-Point Energy function is defining the radius equation. To reiterate, this is the radius of the smallest sphere that passes through one point on the curve while also being tangent to another point on the curve. The approach for determining the radius utilizes the fundamentals of geometry such as line and plane intersections. This approach was taken in an effort to demonstrate the utility of the course material.

Step 1: Find the center of the smallest sphere connecting three points [Figure 6].

- In 2D space, this is done by finding the intersection of the perpendicular bisectors for the two lines that connect points P1 and P2, and P2 and P3.

- In 3D space, the equations of the planes are used:

$$Ax + By + Cz + D = 0$$

$$n_x X + n_y Y + n_z Z - \left(n_x X_{mid} + n_y Y_{mid} + n_z Z_{mid}\right) = 0$$

- A sub-function was created in python to determine the equation of the line where the two planes intersect - this is vertical line passing through the center of the sphere.

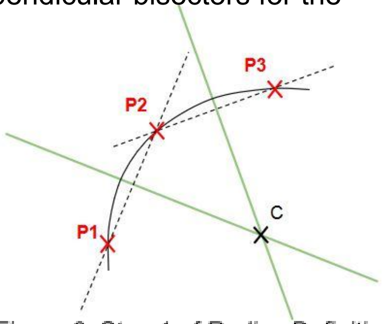


Figure 6: Step 1 of Radius Definition

Step 2: Introduce the tangency condition [Figure 7].

- To introduce the tangency condition of Tangent-Point Energy, move points P1 and P2 immediately next to each other.

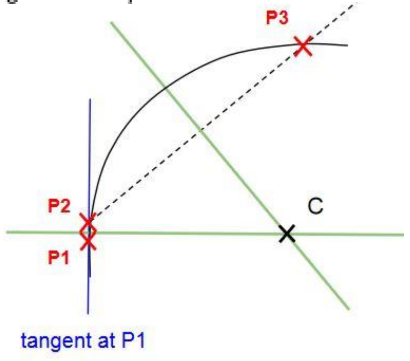


Figure 7: Step 2 of Radius Definition

Step 3: Determine the radius [Figure 8].

- Find the shortest distance from the point P1 that intersects the vertical line passing through the center of the sphere - this is the Tangent-Point Energy radius

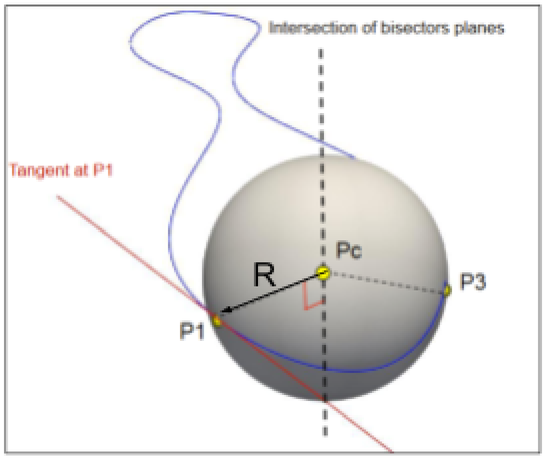


Figure 8: Step 3 of Radius Definition

2.1.2 Identification of Points with Highest Energy

The second step in implementing the Tangent-Point Energy is to identify the points with the highest energy that should be optimized. Before this can happen, the energy for each point along the curve should be calculated using the radius equation previously defined. So, the equation for the Tangent-Point Energy becomes:

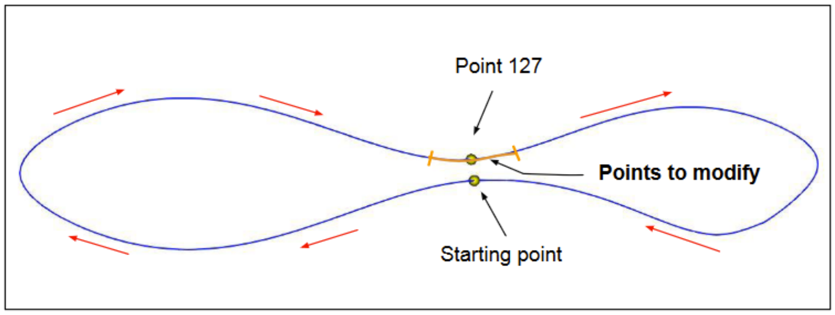
$$\epsilon = \frac{1}{(\sqrt{(x-x_c)^2+(y-y_c)^2+(z-z_c)^2})^\alpha} = \frac{1}{R(x,y)^\alpha}$$

where the subscript c denotes the coordinate of the center of the sphere. The parameter ***α*** was chosen as 2 for this algorithm because Tangent-Point Energy will be Möbius Invariant if it is any less.

Since the goal is to eliminate self-intersections rather than just minimize tangent-point energies (which would result in the curve being a circle), a criteria must be defined for what constitutes the location of self-intersection. Chosen criteria:

High Energy Criteria: *points with energy magnitudes that are at least 60% of the maximum energy calculated for the entire curve.*

This criteria was chosen as it adequately captures the point with highest energy (which would be the point of self-intersection) as well as the neighboring points which may contribute to the self-intersection. Figure 9 demonstrates this criteria as a threshold on the total energy curve.



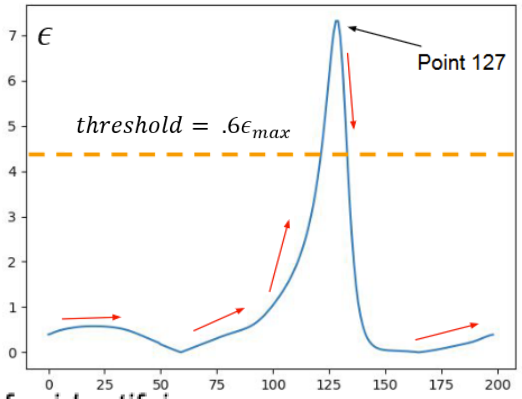


Figure 9: Visualization of the process for identifying points with highest energy

2.2 Implementation of Gradient Descent

Gradient Descent is a rudimentary optimization method that utilizes the gradient function to translate points in the direction of a local minimum. This method is used to minimize the points identified as having highest energy on the curve. It is an iterative method, meaning after each step, the algorithm recalculates the points with highest energy and repeats the translation process.

2.2.1 Development of Gradient Equation

The general equation for Gradient Descent is:

$$p_{i+1} = p_i - \eta \cdot \nabla \epsilon(p_i) \qquad p_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$$

Where $\nabla \epsilon$ denotes the gradient of the Tangent-Point Energy function. The gradient of a function requires the partial derivatives with respect to each coordinate axis:

$$\nabla \epsilon = \frac{\partial \epsilon}{\partial x}\hat{\imath} + \frac{\partial \epsilon}{\partial y}\hat{\jmath} + \frac{\partial \epsilon}{\partial z}\hat{k}$$

For this algorithm, the partial derivatives were taken for the Tangent-Point Energy function and the resulting gradient formulation is shown here:

$$\nabla \epsilon = (\frac{-\alpha(x - x_c)}{((x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2)^{\frac{\alpha}{2}+1}}, \frac{-\alpha(y - y_c)}{((x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2)^{\frac{\alpha}{2}+1}}, \frac{-\alpha(z - z_c)}{((x-x_c)^2 + (y-y_c)^2 + (z-z_c)^2)^{\frac{\alpha}{2}+1}},)$$

2.2.2 Iterative Translation of High Energy Points

A localized approach to Gradient Descent is used for this algorithm, meaning discrete points are moved rather than moving the control coordinates. This approach was chosen because the VTK Parametric Spline function automatically defines the control coordinates for the input curve. Manipulating the curvature of the spline without access to the control coordinates becomes difficult from a global perspective, though local control also presents some challenges as will be described in the Results and Future Work sections.

The local approach involves iteratively translating each discrete point with high energy a stepsize in the opposite direction of the gradient. After each point is moved one step, the algorithm then recalculates which points have the highest energy, and then the process is repeated. Points are translated towards lower energies until the algorithm reaches the defined iteration limit. Because the algorithm recalculates the gradient and energy function every iteration, increasing the number of iterations means prolonging the computational time dramatically. The limit chosen for this algorithm was 20 iterations, which equates to around two minutes of calculations.

An important consideration for our implementation of Gradient Descent is the choice of learning rate, which is denoted by η in the Gradient Descent equation depicted in the previous section. The learning rate defines the size of the step a point will be moved in the direction of lower energy, and has a significant impact on whether the optimization algorithm converges or diverges. Too small of a learning rate will significantly increase computational time, while too large of a learning rate will cause the path to bounce around and diverge. Through trial and error tests, we arrived at a learning rate of η = .02.

3 Results

The designed algorithm still requires considerable development to achieve the level of functionality envisioned for the project objective. The current state of the algorithm satisfactorily implements the Tangent-Point Energy function and calculates the energy sphere for all discrete points along the test case spline curve. It also is capable of correctly identifying the location of points with the highest energies. The implementation of Gradient Descent is mathematically correct, however this function is presenting challenges after a few iterations because it begins translating target points in directions that differ from what intuition tells us. Significant troubleshooting efforts have been conducted, though computational time has limited the ability to completely correct the algorithm.

Shown in Figure 10 below is a visualization of a Tangent-Point Energy sphere for two points along the test case curve.

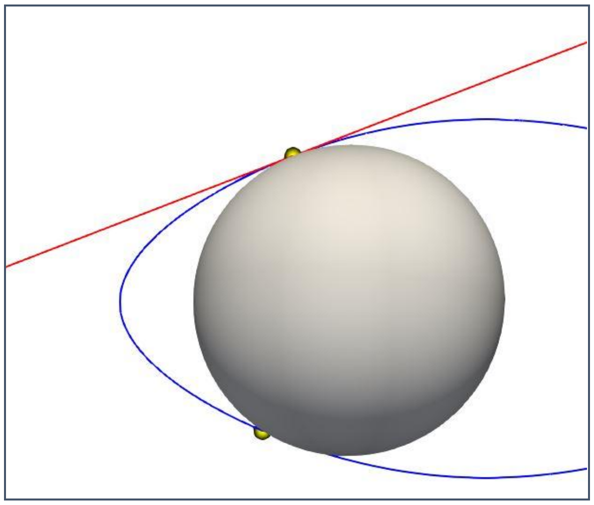
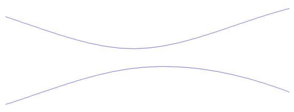
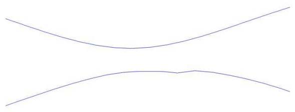
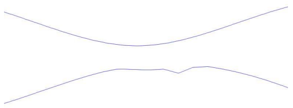
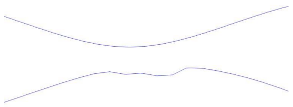
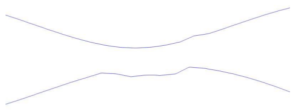
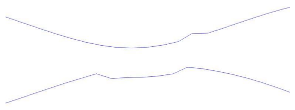
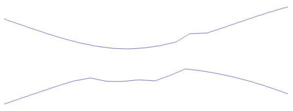
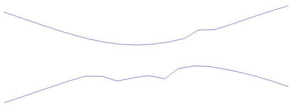
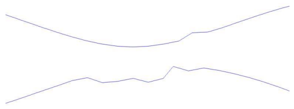
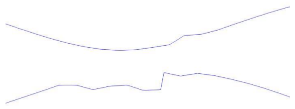
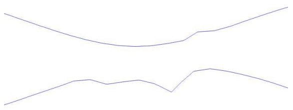
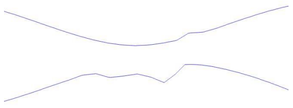
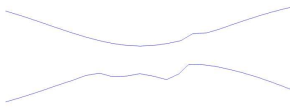
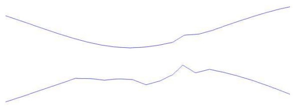
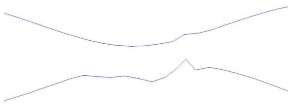
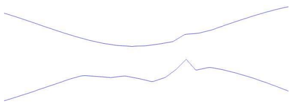
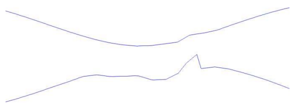
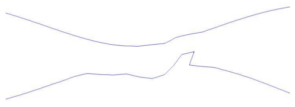
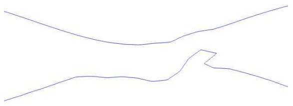


Figure 10: Visualization of Tangent-Point Energy Sphere

The current implementation of the Gradient Descent method is presenting some challenges, however. The behavior of the curve is rather chaotic and not what one would expect when using such a simple spline curve. Table 1 depicts a 20 iteration simulation on a 100-point curve with α = 2 and η = .02. What is interesting about this simulation is that it appears to follow the rules of repulsive optimization. A key example of this is seen in the transition between Step 3 and Step 4 – as the kink in the curve on the bottom-right grows upwards, the opposite side of the curve moves to avoid being too close. This is evidence that the algorithm is attempting to repel points with too high of an energy, though it is very clear that the direction in which it does so is incorrect.

Table 1: 20 Iteration Functionality Test

 Initial Curve	 Step 1	 Step 2	 Step 3
 Step 4	 Step 5	 Step 6	 Step 7
 Step 8	 Step 9	 Step 10	 Step 11
 Step 12	 Step 13	 Step 14	 Step 15
 Step 16	 Step 17	 Step 18	`

4 Summary and Future Work

To summarize, Repulsive Shape Optimization methods attempt to solve an optimal control problem while preventing curve and surface self-intersections. These repulsive methods are particularly adept at modeling geometries found in nature as things in nature must form in such a way that prevents growing through itself. The ultimate motivation for this project is to model a bicep muscle tissue strand, demonstrating the repulsive optimization's ability to accurately model natural geometries. Due to limitations and troubleshooting challenges, that goal could not be fully achieved. However, the algorithm developed for this project forms the basis for this goal as it implements the repulsive Tangent-Point Energy function and uses Gradient Descent optimization to minimize curve energies. Despite successes in implementing the repulsive energy function, there still exist some challenges that need to be overcome with Gradient Descent.

4.1 Implementation Challenges and Limitations

As of now, the energy calculation seems to give correct results, even though a different approach could be used in order to reduce the calculation time. Indeed, for every iteration of the optimization process, the energy of every point with respect to every other point is calculated which can be very time consequential. There exist optimization acceleration methods that work to reduce the calculation time by discretizing the inner product of the integral, though these methods are typically very complicated and outside our realm of academic understanding.However, by reducing this calculation time, a simulation with more iterations and a smaller learning rate could be done which might give better results. It could also be possible to create a curve with a lot more elements than 200 and increase the accuracy of the program.

Another challenge that needs to be overcome is the way the gradient descent is applied. Currently, local nodes are modified at every iteration if they possess a really high energy. This local aspect of the optimization might be one of the reasons for the problems encountered. Individually translating discrete points on the curve inherently means each point will follow the direction of its own gradient, which can often lead to points translating in odd directions. One consideration that would mitigate this error is using a global approach to the optimization. By using the control coordinates of the spline to determine the overall geometry, the direction of optimization can be controlled more smoothly and prevent the wonky iterations seen in the local approach. This would require using a different python toolkit function to generate the input spline. However, once that is done, the Gradient Descent process would remain exactly the same – calculate the points with highest energy, shift the control points so that the discrete points move in the direction of decreasing energy, and then repeat.

4.3 Future Work

Immediate attention is required to correct the errors present with the Gradient Descent function. The most promising plan of action would be to create the input spline in a different way where the control points could be used to have a global influence on the shape of the spline. By doing so, these control points could be modified with the gradient descent method instead of the local nodes that are currently used.

The desired outcome of this entire endeavor is to demonstrate the real-world utility of Repulsive Optimization by modeling a bicep muscle strand. This problem could be formulated as multiple spline curves, each with two static control points in the place of where the shoulder ball-and-socket joint and elbow joint are. One consideration to develop in the future would be the interaction of Tangent-Point Energies between multiple curves, rather than just with respect to a single curve. The process would be similar, though computing energies between several curves could be extremely expensive.

Ultimately the application of these theories is very pertinent to many industries, and the development of such a tool could progress 3D modeling methods in a very interesting way.

References

[1] *Optimal Control Problem*, http://liberzon.csl.illinois.edu/teaching/cvoc/node4.html.

Andnicolevorderobermeier Arxiv:2104.10238v1 [Math.AP] 20 Apr 2021. https://arxiv.org/pdf/2104.10238.pdf.

[2] “Imaging Maths - inside the Klein Bottle.” Plus Maths, 1 Sept. 2003, ttps://plus.maths.org/content/imaging-maths-inside-klein-bottle-15.

[3] Keenan Crane - Repulsive Surfaces, https://www.cs.cmu.edu/~kmcrane/Projects/RepulsiveSurfaces/index.html.

[4] Saslow, Wayne M. “Coulomb's Law for Static Electricity, Principle of Superposition.” Electricity, Magnetism, and Light, Academic Press, 9 May 2007, https://www.sciencedirect.com/science/article/pii/B9780126194555500024.

[5] O’Hara, Jun. “Energy of a Knot.” Topology, Pergamon, 21 Mar. 2002, https://www.sciencedirect.com/science/article/pii/0040938391900102.

[6] Keenan Crane - Repulsive Surfaces, https://www.cs.cmu.edu/~kmcrane/Projects/RepulsiveSurfaces/index.html.

[7] Symmetry, Integrability and Geometry: Methods and ... - Massey University. https://www.massey.ac.nz/~rmclachl/sigma16-080.pdf.

[8] Alonso, Gustavo, et al. “Optimization Methods.” Desalination in Nuclear Power Plants, Woodhead Publishing, 1 May 2020,https://www.sciencedirect.com/science/article/pii/B9780128200216000053.

[9] Brownlee, Jason. “Gradient Descent with Adadelta from Scratch.” *Machine Learning Mastery*, 11 Oct. 2021, https://machinelearningmastery.com/gradient-descent-with-adadelta-from-scratch/#:~:text=Gradient%20descent%20is%20an%20optimization%20algorithm%20that%20uses%20the%20gradient,of%20partial%20derivatives%2C%20called%20Adadelta.

[10] Kwiatkowski, Robert. “Gradient Descent Algorithm‐a Deep Dive.” *Medium*, Towards Data Science, 24 May 2021, https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21.

[11] Real Python. “Stochastic Gradient Descent Algorithm with Python and NumPy.” *Real Python*, Real Python, 19 Jan. 2021, https://realpython.com/gradient-descent-algorithm-python/.

[12] Roshchupkin4, S P, et al. “IOPscience.” *New Journal of Physics*, IOP Publishing, 22 Dec. 2021, https://iopscience.iop.org/article/10.1088/1367-2630/ac46e3.

[13] Saeed, Mehreen. “Gradient Descent in Python: Implementation and Theory.” *Stack Abuse*, Stack Abuse, 4 Nov. 2020, https://stackabuse.com/gradient-descent-in-python-implementation-and-theory/.

[14] Sebastian Ruder. “An Overview of Gradient Descent Optimization Algorithms.” *Sebastian Ruder*, Sebastian Ruder, 20 Mar. 2020, https://ruder.io/optimizing-gradient-descent/.