Using Unity and Virtual Reality to Demonstrate Knot Energies

Alexander Curtis

April 2019

1 Introduction

The intent of this semester's project was to create an application that would allow for users to interact with knots in virtual reality. The long term goal is implement a function that will minimize the energy of a knot and compare the ideal calculation to some user's attempt to untangle the knot. The progress made so far allows for an example knot to be read in from a data set, then drawn, and then finally its untangled iterations read in via a JSON formatting package. The future goals of the project will be; to create an intuitive system to allow users to click on points and adjust the position of the knot, implement an energy calculation detailed by Sören Bartels and Phillip Reiter [1], create a comparison function that will compare the users' solution to the optimum minimum energy calculated, and create a video game like interface for user friendliness. The larger objective and scope of the project is to allow interactions with abstract mathematical objects in virtual reality so that users can have a tangible way of working with mathematical concepts.

2 Knot Energies

Our working definition of a knot is any closed non-self-intersecting curve within 3D space. Multiple energy functions have been defined such as the Möbius energy and O'Hara energy [2]. For this project, we use the Tangential Point Energy for its property of avoiding self-intersections. By avoiding self-intersections, we mean the function blowing up when an intersection occurs. We note the tangent point functional as it is a substantial portion of our Energy measurement. The tangent-point radius $r(u(y),\ u(x))$ is the radius of circle that is tangent to u at $u(y)$ and then intersects with u in $u(x)$. For a parameterized curve u we have

$$[1] \qquad u \mapsto \frac{1}{2^q q} \iint_{I\times I} \frac{dxdy}{r(u(y),u(x))^q} \qquad q > 2$$

With this in mind, the functional $TP(u)$ where $u : \mathbb{R}/\mathbb{Z} \to \mathbb{R}$ is continuously

differentiable, embedded, and regular then can be constructed such that:

$$TP(u) = \frac{1}{q}\iint_{\mathbb{R}/\mathbb{Z}\times\mathbb{R}/\mathbb{Z}} \frac{|u'(y) \wedge (u(x) - u(y))|^q}{|u(x) - u(y)|^{2q}} dxdy \qquad q > 2$$

By $\wedge$, we mean the vector product. Finally, we define the energy of a given knot to be,

$$E(u) = \frac{\varkappa}{2}\int_I |u''(x)|^2 dx + \varrho TP(u)$$

where $\varkappa$ and ϱ are chosen positive parameters. The choice of $\varkappa$ and ϱ will affect numeric computation and the stability of the calculation. Minimizing this functional is the primary objective as it will 'untangle' a given elastic knot provided proper parameters have been chosen. Examples of successful parameters can be found in Sören Bartels' and Philipp Reiter's paper. In addition, the time-stepping method and gradient flow descriptions to compute this can be found in detail as well.

3 Unity

The unity engine is a game engine developed in 2005 by Unity Technologies. It's API is implemented in C#. It was chosen to develop this project because of it's fairly liberal copyright policies allowing individuals and educational institutions to freely create projects. In addition, C# has plenty of resources available online for free via Microsoft's own website or countless third party forums. The VR portion of this project uses the Oculus Rift which readily works with Unity after a couple of simple package installations.

4 Bézier Curve Introduction

The first method of drawing a desired curve failed because of an attempt to overcomplicate a process already done. The process of drawing a curve is already completed if we can choose discrete points with sufficient density to create a curved line in space. However, this dead end did provide a lot of interesting math and experience with the Unity that is worth mentioning.

A Bézier curve is a parametric curve comprised of a polynomial on some parameter t which provides control points, allowing us to push and pull a curve to a desired state. Connecting these curves will give us a spline. Closing the spline can give us a knot (provided this closing process does not deviate from the definition of a knot). The issue with this method was if we were given an initial data set with a large number of points, a couple of questions arise that were not immediately relevant to the project and proved to be quite difficult. How do we define a Bézier curve which connects a given set of points? How many curves with control points do we allow? If we have too many Bézier curves, we could produces large spikes sticking out on the curve between two relatively close points. From a user's perspective, this makes the process of minimizing the

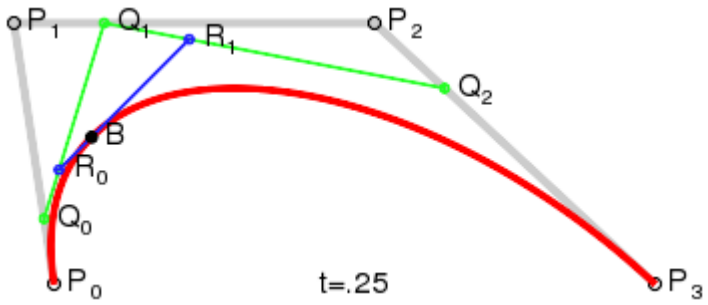
energy of the knot nightmarishly tedious as the numerous minuet adjustments would cause most rational people to give up. In addition, Bézier curves have some limitations, such as their approximation of the circle. This can only be done with a certain number of curves and respectively a certain degree Bézier curve. This would restrict the flexibility of the closed spline and limit the number of orientations possible. These complications lead to this method being abandoned as the provided data was dense enough to produce a curve from a set of points.

4.1 Bézier Curves

Definition 4.1 (Bézier Curve)**.** We may explicitly define a Bézier Curve of degree n as

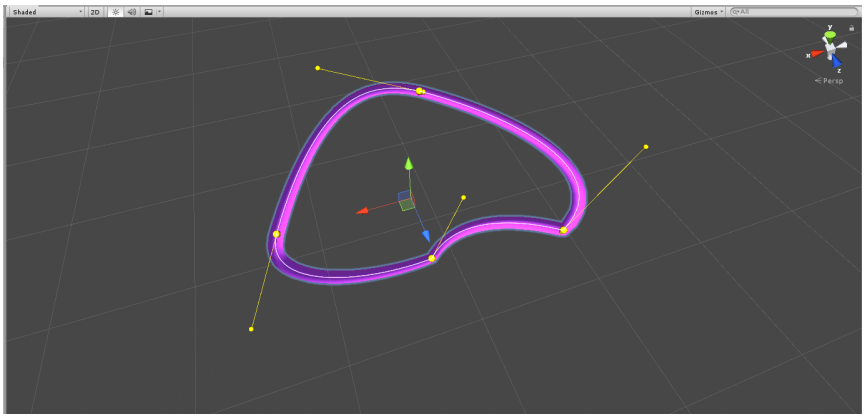
$$[4] \qquad \mathbf{B}(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{n-i} t^i \mathbf{P}_i$$

Note that $\mathbf{P}_1$ and $\mathbf{P}_n$ are the start and end point for the desired curve. $\mathbf{P_i}$ such that $1 < i < n$ are control points that typically do not lie on the curve. Their significance is that by adjusting their position we may push and pull our curve in space.



*Cubic Bézier Curve by Chris*828, *Public Domain* [4]

In this diagram the line segment $\overline{R_0R_1}$ is actually responsible for drawing our red curve. As $\overline{R_0R_1}$ traverses $\overline{Q_0Q_1Q_2}$, this segment is pivoted across $\overline{P_0P_1P_2P_3}$. This motion describes a curve in space. Extending this to three dimensions and using Unity, we can represent a spline and close it to form a knot. In this image we are using four Bézier curves to construct a deformed closed circle. The yellow orbs projected outside of the spline are the control points that will twist and morph the object.The image below is generated by a package provided for free on the unity store that allows one to draw closed splines and other curved 3D objects. As previously mentioned, this method did not work out as originally intended but did lead to a solid familiarity with the unity engine. The functions used in instantiating the curve object were used later on to create a knot by a different method.

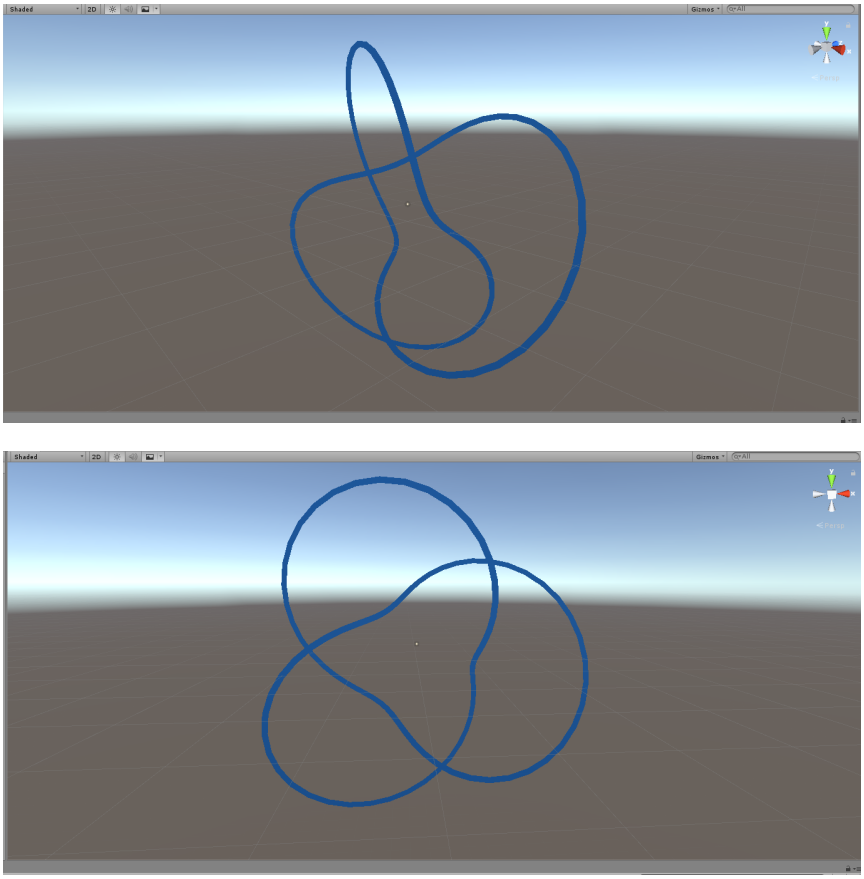


5 JSON formatting and Drawing a Knot

The first major hurdle regarding interpreting data was reading the data provided by Phillip Reiter as a JSON format document. JSON (JavaScript Object Notation) is a format that provides a standardization of data-sets and objects using javascript syntax.[3] Unity by default does not possess such a feature so a Unity version of Newtonsoft JSON framework was a necessary addition. After installing this package, we can read and write data freely and legibly using a standard file format. This makes exchanging data or editing data far more efficient. However, the default package manager for Visual Studio does not cooperate with this package and it must be added manually. As an example, here is a basic read function that imports 3D coordinates from a file to a List class.

```
public static List<Vector3> ExampleDataSetRead() {
       string data = ReadString("Assets/Resources/CurveA.txt");
       List<Vector3> returnList = new List<Vector3>();
       JObject dataJObject = JObject.Parse(data); //read the dataset
           string held in text
       JEnumerable<JToken> position = dataJObject["pos"].Children();
           //LIST all positions
       foreach (JToken pos in position) //iterate and assign x y z to
           VECTOR3 in unity
       {
           Vector3 vertex = new Vector3(); //multiply values by 10 for
               sufficient distance in Unity's space
           vertex.x = pos.Value<float>("x")*10;
           vertex.y = pos.Value<float>("y") * 10;
           vertex.z = pos.Value<float>("z") * 10;
           returnList.Add(vertex);
       }
       return returnList;
   }
```

We can read the entire data just as a large string as it is small enough to do so. Then using Newtonsoft's JObject and JEnumerable class, we can read each value in our JSON array to a Vector3 object (a class to represent 3D points in unity). Using this list we can draw a line connecting these points in a discrete manner. The density of data is sufficient enough that the 'illusion' of a curve remains as demonstrated below.



After, we have drawn our initial state we must parse through the iterations of (x, y, z) coordinates of the various iterations that would be produced by our energy reducing function. To demonstrate this, we use a flavor of the function previously mentioned to read in JSON data and bring it into memory. This is a rather large amount of data and for this example knot it is approximately 30 MB of data. We create a List of Lists using the List class provided in the generic collection of C# library to organize our data. The inner list contains the coordinates for a given state of the knot. The outer list contains all states of the knot. For this example, we have around 2000 iterations where the minimizing algorithm has reduced the energy of the knot. With such a high number of

iterations, we can use an update function that redraws our knot into the next iteration for each frame respectively. The result, hypothetically, should be a smooth animation smoothing out the knot. Currently, this function has a bug that will be fixed in the immediate future.

6 Concluding Remarks

The experience of working on a relatively new problem in math was extremely interesting. The project appears to have made some progress and with further development can become something fruitful in terms of math pedagogy. Using Unity and virtual reality in tandem can create interesting experiences for users and it potentially allows for math research to become much more accessible to the layman. The technology can aid interest in mathematics by creating a compelling interactive experience and perhaps can aid in the education process as well. There is a lot to be explored and a very untapped resource is at hand for mathematicians.

References

[1] Bartels, S. & Reiter, P. Stability Of A simple Scheme For The Approximation of Elastic Knots And Self-Avoiding Inextensible Curves. 2018, arXiv e-prints , arXiv:1804.02206. https://arxiv.org/abs/1804.02206

[2] Blatt, S. The Gradient Flow of O'Hara's Knot Energies. 2018 April, Mathematishe Annalen. https://link.springer.com/article/10.1007/s00208-017-1540-4

[3] Introducing JSON. https://www.json.org/

[4] Wikipedia contributors. 2019, April. Bézier curve. Wikipedia, The Free Encyclopedia.