Polygons and Potential Geodesics in 2 & 3 Dimensions

Michael Glover

To begin, I will briefly explain exactly what it is that I completed in my weeks of research this summer. My experiences ranged from relatively simple calculations, such as calculating dot and cross products of vectors, to calculating partial derivatives and determinants of matrices, while also utilizing my newfound knowledge in second order differential equations.

Initially, I began with practicing with a supplied code that modeled a 3-gon changing through time, with the primary objective of conserving its area. This code, when provided with initial velocity and positional values for each the vertices of the 3-gon, would plot the initial shape created as well as the final shape as it had changed with the given velocities through time; an example can be seen below in Figure 1.

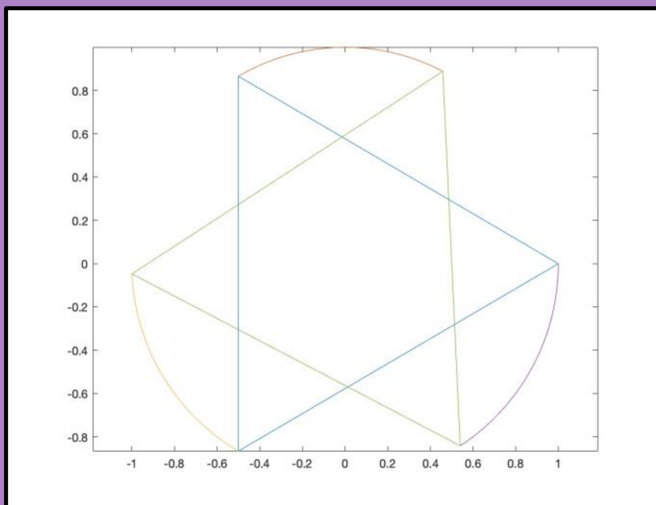

Figure 1. A still image of an initial 3-gon undergoing a basic rotation about its center. This displays an output of the code I was provided without any modifications.

This point was where I began my modifications to the code I was provided. First, instead of only providing a still image of the transformation of the 3-gon (those being the initial shape and final shape), I wanted the code to create an animation of the 3-gon as it changed through time. This change presented a model that provided an easy and visually appealing way to plot these shapes through time, as now the shape can be seen at each time iteration as determined by the differential equation tool provided within MATLAB.

After the animation was introduced into the code, I decided to move the focus of my efforts into creating a new script that could handle and animate the same information, but now for 4-gons. This new animation can be seen in Figure 2.

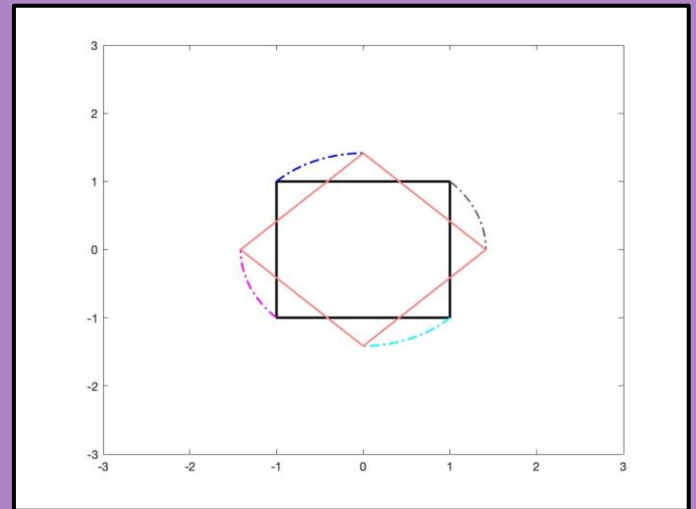

Figure 2: The new adapted code displaying an animation of a 4-gon rotating about its axis, which is like the 3-gon's movement in Figure 1.

This modification took a substantial amount of time, but this allowed for new possibilities that I didn't initially think about; this included the modeling of degenerate 4-gons. These are 4-gons in which two of the vertices lay on top of one another, resembling a 3-gon in a 2D space. This led us to inquire about whether a degenerative 4-gon and a regular 3-gon would result in the same path traveled, when applying the same initial position and velocity to them both. I created a code that, essentially, stacked the two previous animations on top of one another. This allowed us to see how the two shapes differ in their movement, as well as the exact moment in which their paths begin to diverge. The new code that includes the plotting of each of the independent shapes on top of one another can be seen in Figure 3.

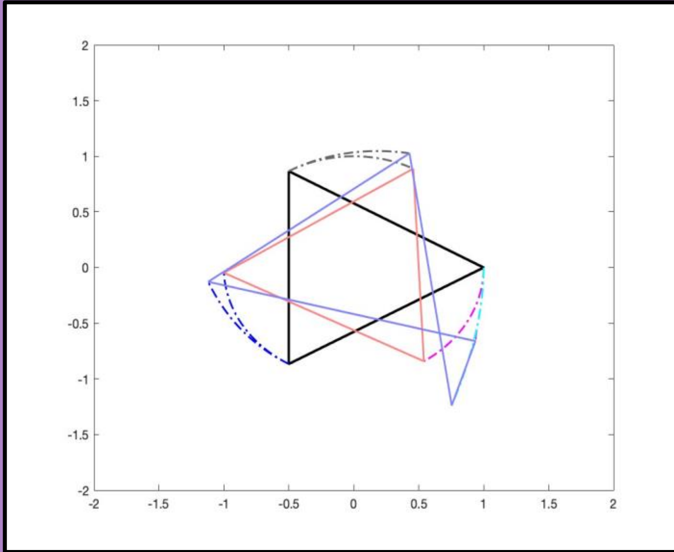

Figure 3. This shows a 3-gon and a degenerative 4-gon. The last vertex of the 4-gon was copied from the 3rd to demonstrate the degenerative qualities. Both shapes received the same initial velocities, and their travel through time is modeled as such.

This was code adaptation was a feat for me, requiring new knowledge of complex topics I had yet to come across in my academic career thus far. After tackling 3-gons and 4-gons, I turned my attention to applying the same concepts to tetrahedra. The code only worked currently in 2D, so transforming it to work in a 3rd dimension was tricky. I wrote the code such that the initial qualities of the fourth vertex were dependent upon the other three. This is demonstrated below, where V refers to the initial velocities and A refers to the initial positions.

$$V_4 = -(V_1+V_2+V_3)$$

$$A_4 = -(A_1+A_2+A_3)$$

Having the fourth vertex be dependent allowed for the code to be simplified greatly, letting me progress faster than I anticipated. The tetrahedra code animates an initial tetrahedron through time, tracking the positions of its vertices as it goes to calculate the volume generated by them. This allows us to model the behavior of these tetrahedra, as well as determining if a specific combination of positions and velocities result in a volume that is conserved. Figure 4 demonstrates the animation of the tetrahedron through time.

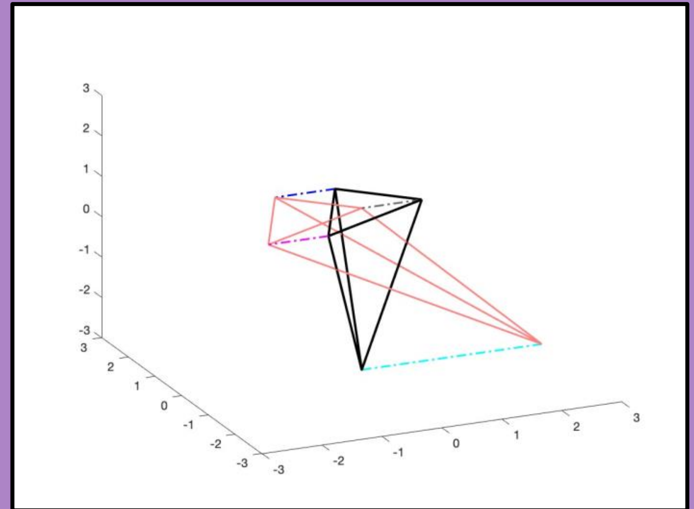

Figure 4. This demonstrates a tetrahedron undergoing a volume conserving movement. The initial shape can be seen in black, and the final, in orange. The paths of each of the vertices can be seen by the dotted/dashed line.

This was a big achievement for me in my research, probably the biggest yet. We ran into one problem, however. The code worked fine in certain scenarios, but sometimes, the computer would introduce a small level of computational error. This error compounds as the code runs, reaction resulting in a more and more inaccurate result. It would be a goal to have this inaccuracy diminished, but I believe I have run out of time to tackle it.

This isn't all I focused on, however. One of my main goals is maintaining area and volume conservation through the manipulation of the initial conditions of the vertices of the respective shape we are discussing. I developed a code, that when given a set of initial positions and initial velocities, can calculate whether that combination will result in the area or volume being conserved. If that set of values doesn't result in a conservation, then a projection is calculated and provided to the user. This projection is a suggested initial velocity vector, that when combined with the original position vector, should result in a conservational transformation. The expression for calculating the projection field can be seen below.

$$W \rightarrow W - \frac{(\nabla V_\gamma \cdot W)}{\left\|\nabla V_\gamma\right\|^2} * \nabla V_\gamma$$

In this expression, W represents the initial velocity vector inputted by the user. ∇V_γ is the gradient of the position vector at any given time, γ.

For the combination of positions and velocities to result in a conservation of area, the difference between W and the right side of the expression needs to equal zero. If it doesn't, the difference between the two is the new projection velocity to be considered for conservation. My code detects this value, and when it equals zero, displays, "You've got it!" as a confirmation of the user's success. When the difference isn't zero, it provides this value to the user in an easy-to-read fashion, so that it can be inputted back into the code to check its validity and accuracy. An example of the formatting can be seen in figure 5 where it is applied to tetrahedra, but I also developed the same code to work with 3-gons and 4-gons.

```
>> doesitwork
You Got It!
>> doesitwork
Try These Instead
V1: -1.2        0.9        -0.1
V2: 1       -0.8       -0.2
V3: 0.8        1.1        0.1
```

Figure 5. This is an example of the output of my code for tetrahedra. On the first run, the code calculated the volume of the tetrahedron would be conserved, thus outputting the confirmation phrase. After this, I manually changed one of the velocity values to something that I knew wouldn't work. This resulted in my code outputting the calculated projection field based upon the differences discussed earlier, with each column referring to the x, y, and z dimensions.

After creating a script that would determine whether a conservational result would be achieved through any given initial position and speed conditions, I turned my attention, again, towards modeling these shapes through time. This time, however, I wanted to work in three dimensions, using time as the unit for the z-axis. This allowed for a three-dimensional representation of a two-dimensional shape through time, again, using time itself as the unit along the z-axis. As I am only being concerned with velocity and position combinations that conserve area through time, I relied heavily on my previous code to guide me.

My "Does it Work?" code allowed me to see if combinations resulted in the outcome I was searching for, which I would then plug into my first code to model it in two-dimensions through time. I had to modify this code, however, to output the positions of each of the vertices of the given shape at each given time step. This provided me with the coordinates of each of the vertices at each iteration of the model its position through time. I was provided another OpenSCAD script by Dr. Taalman which I utilized to create the three-dimensional models that I would later 3D print. I was able to modify both my 3-gon and 4-gon code to output the positions of their respective vertices.

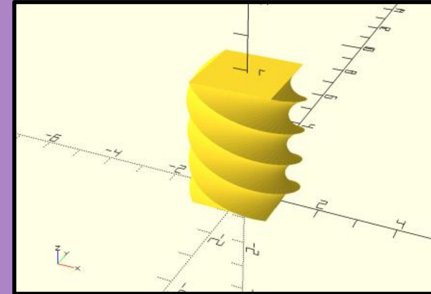

Figure 6. This displays the image generated from the preview of the OpenSCAD code. This shows the given the positional information of a square's vertices, using time on the Z axis. The mesh generated here can be outputted as an .STL file, which can be further utilized in different modeling software.

Now that my two-dimensional modeling codes were able to output the positions of the vertices, I went to work on bringing these three-dimensional models into fruition. I was able to utilize Dr. Taalman's OpenSCAD code to model both 3-gons and 4-gons through time. My two-dimensional modeling code had some limitations in the way it presented the information, however, as I was only able to get the real time positions for each of the vertices in the X and Y dimensions. Because we are introducing a new dimension into the output, there isn't a real time value within the code itself to place in the Z coordinate's place. Thus, there is just a placeholder value, 1, in every Z position for each vertex's position. I had to manually go in, replace the 1s in the first line with 0s, replace the 1s in the third line with 2s, and so on until there were no more two-dimensional layers to match up with a sliced time value. This was time consuming, and If I had the chance, would be something I would try to improve in the future. It got the job done, but getting the output of my code formatted in just the right way can save lots of time when copying it into the OpenSCAD script. The main objective of this script is to take the layered positions of the given shape and connects them together using a mesh. This creates a smooth surface on the 3D model instead of a jagged, stair-like face.

I used all three of these codes to generate many very fascinating models for both 3-gons and 4-gons. They are included on the next page, with labels describing each of them and their respective movements through time.