Heisenberg VR Project Report

Brian Popeck

April 30, 2020

Contents

1 Introduction

1.1 The Heisenberg Group

The Heisenberg group equips $\mathbb{R}^3$ with a particular metric structure. Curves belonging to the Heisenberg group often have a non-intuitive structure. In particular, geodesics, or shortest paths, between points in the Heisenberg group differ from Euclidean lines in intriguing ways. The Heisenberg group has remained an active area of research (including contributions from multiple researchers at Pitt) with many longstanding open questions.

In the Heisenberg group, for a given point $p = (x, y, z)$ in $\mathbb{R}^3$, the tangent at point p to a *horizontal* curve containing p belongs to a two-dimensional vector space (*horizontal* space). From the definition of the Heisenberg group, this vector space contains all vectors v satisfying the relation $0 = v_3 + 2(xv_2 - yv_1)$ and has basis vectors $X = (1, 0, 2y)$ and $Y = (0, 1, -2x)$. Differentiating both sides then integrating from 0 to t, we find that v_3 is completely determined by the curve's starting position and projection into the xy-plane: $v_3(t) = v_3(0) - \int_0^t 2(v_1(s)v_2'(s) - v_2(s)v_1'(s))ds$.

1.2 Project Aims

This project seeks to employ virtual reality to support visualizations of the Heisenberg group for learning purposes. To support more advanced visualizations, we created a script to plot a geodesic between any two points in space. This plotting of arbitrary geodesics became our first visualization. The second visualization is a driving game which restricts the driver to only move in horizontal directions at a particular point in space. Our third and final visualization is a program which connects vertices and their midpoints with geodesics in a recursive manner in order to investigate the Gromov conjecture [1] as it relates to surface-filling in the Heisenberg group. All three visualization were implemented with the Unity game engine to simplify the rendering of curves.

2 Drawing Geodesics

This program asks the user to input two points in $\mathbb{R}^3$, and then draws the geodesic between the two given points. The script accomplishes this by applying a set of transformations to the given endpoints p_1 and p_2, using a given formula for a particular set of geodesics, then applying the inverse of the initial transformations to the geodesic which restores the endpoints to their original position.

Importantly, the set of applied transformations are isometries which preserve the geodesic nature of a translated curve.

2.1 Isometries in the Heisenberg Group

As explained on Dr. Schikorra's webpage [2], we have the isometries of scaling, translation, and rotation. For a given point $p = (x, y, z)$, these transformations are:

- Scaling: The point p scaled by a positive scalar r is given by $(rx, ry, r^2 z)$.
- Rotation: Let α be a counterclockwise rotation in the xy-plane. The rotated point is given by

$$\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

- Translation: Let $q = (x', y', z') \in \mathbb{R}^3$. The point p translated by q is given by $(x + x', y + y', z + z' + 2(x'y - xy'))$.

The program applies these isometries to the given endpoints so that one transformed endpoint is at the origin $(0, 0, 0)$. More precisely, both endpoints are translated by $-p_1$ which moves p_1 to the origin. Note that if the z-component of p_2 is negative after this translation, we swap the initial values of p_1 and p_2 and then perform the translation again. By the translation formula, this swap essentially multiplies the transformed z-component by -1, guaranteeing that p_2 has a non-negative z-component after translation.

Once p_1 has been translated to the origin, we have three cases for the remaining transformations to apply, depending on the result of transforming p_2.

In the first and simplest case, if the z-component of p_2 is equal to 0, then the geodesic between p_1 and p_2 is the Euclidean line segment connecting them. To find our original geodesic, we simply translate a set of points on the geodesic (to be rendered) by p_1, the inverse of our one and only transformation applied in the beginning.

In the second case, if the x-component and the y-component of p_2 are each equal to zero, then we directly apply the following formula from Dr. Schikorra's webpage [2] for a geodesic from the origin to a point on the z-axis:

$$\gamma(t) = (1 - \cos t, \sin t, 2(t - \sin t)) \quad \forall\, t \in [0, 2\pi]$$

.

We similarly find our original geodesic by translating a sufficient set of points by p_1, the inverse of the only transformation applied.

Finally, in the third and most common case when neither of the above conditions hold, the other transformed endpoint is at (x, y, T) for some $x, y \in \mathbb{R}$ and $T \in \mathbb{R}$ with $T \geq 0$.

In this case, both transformed endpoints are scaled and rotated until p_2 is of the form $(1, 0, T)$ for some $T \geq 0$. Note that neither scaling nor rotation moves p_1 from the origin, and neither scaling nor rotation can cause a sign change in

the z-component of p_2, guaranteeing $T \geq 0$ after scaling and rotation. Thus, in practice these two transformations need be computed on p_2 only.

First, p_2 is rotated so that its y-component is equal to 0. The angle of rotation may be found by calculating the arc-tangent of p_2 in the xy-plane. Afterwards, p_2 is scaled by R, the reciprocal of its x-component, such that after this transformation p_2 is of the form $(1, 0, T)$ for some $T \geq 0$.

At this stage, we apply an explicit parameterization of the unique geodesic from $(0, 0, 0)$ to $(1, 0, T)$ as derived on Dr. Schikorra's webpage [2]. The parameterization is as follows, where $\eta(t)$ is a geodesic from the origin to $(0, 0, 4R^2)$ of the form described in the second case and s satisfies $\frac{s-\sin s}{1-\cos s} = T$:

$$\gamma(t) = \begin{bmatrix} \frac{1}{2R} & \frac{T}{2R} - sR & 0 \\ -\frac{T}{2R} + sR & \frac{1}{2R} & 0 \\ 0 & 0 & 1 \end{bmatrix} \eta(t), \quad t \in [0, s].$$

Then, we take a set of points from this geodesic, scale by $\frac{1}{R}$, rotate by the opposite angle of our initial rotation, and translate by p_1 to return the endpoints to their starting positions and render the desired geodesic.

2.2 Computational Details

For a clearer visualization, the "horizontal" axis in the program was chosen to be the y-axis, which in Unity appears vertical to the user when using the default camera orientation. In the notation used in this report, we take the z-axis to be horizontal, so in the implementation the y-component and z-component are swapped.

In the third case, the explicit parameterization used is a function of s, the solution to $f(s) := \frac{s-\sin s}{1-\cos s} = T$ for a given T. $f(s)$ is monotonic for $T \in [0, 2\pi)$, so our program estimates s by computing T for one thousand equally-spaced values of s in the interval $[0, 2\pi)$ exactly once at the start of the program. Then, when a user asks to plot a geodesic, the program takes the value of T and looks in the stored list of T values for the closest match, and the s value used to calculate that closest match is then used for the remainder of calculating the geodesic. This approach has the advantage of computational efficiency, since finding s for each geodesic becomes a constant-time lookup during sampling, but due to this estimate, s may have some error, particularly for large T where the difference between adjacent saved T values may be large.

The parameterization of the desired geodesic, calculated as explained above in each of the three cases, is implemented as a function created dynamically at runtime. When a new Geodesic object is constructed in the program, the parameterization function is created exactly once dependent on the supplied endpoints. For the remainder of the lifetime of the Geodesic object, this parameterization function may be sampled to obtain points on the geodesic for rendering or any other purpose. This approach enables the correct tranformation routine to be set exactly once depending on which of the three cases apply.

The geodesic is rendered via the LineRenderer object in Unity, which takes a set of points as input. This object was chosen since our implementation can easily sample points along the geodesic but does not necessarily contain an explicit parameterization.

2.3 Virtual Reality Integration

The project supports the use of an Oculus Rift virtual reality headset to allow the user to have the camera follow the motions of their head and to be able to fly around in 3D space using the Oculus touch controllers. For this program, the controller may be used to zoom in and zoom out along the direction that the user is currently looking.

The VR support enables a much more natural and intuitive exploration of 3D space, making it easier for mathematicians to use this visualization tool. Note that our program may also be used without the virtual reality headset, with zooming performed by the arrow keys and camera rotation performed via mouse.

3 Driving Game

This program places the user in control of driving a sphere in 3D space. The user provides 2D input from either the arrow keys on the keyboard or the joysticks on the Oculus touch controllers in order to steer the sphere, whose instantaneous velocity is limited to the horizontal space at every point.

On every frame, the 2D input is projected into the horizontal space, and the sphere moves a distance along the projection proportional to the configured velocity of the sphere. In this manner, the path of the sphere approximates a curve in the Heisenberg group.

To enhance the visualization of the curve traced out by the sphere's movement, several features have been added. Most directly, the path traced by the sphere is rendered and colored along a gradient with respect to time. In addition, the program can be configured to trace the projection of the movement of the sphere in the xy-plane.

Finally, the main yellow sphere is intersected at right angles by two rows of smaller, pink spheres which mark the basis vectors of the horizontal space at the given point. Recall that these basis vectors have the form $X = (1, 0, 2y)$ and $Y = (0, 1, -2x)$. The program positions several smaller pink spheres along these vectors in both the positive and negative directions.

The purpose of the game is to provide a mathematician or other interested user with an intuitive sense of how movement can occur in certain areas of the Heisenberg group. The virtual reality controls enable this program to be accessible and facilitate rapid experimentation.

4 Geodesic Triangles

Our third and final visualization builds upon the first visualization of drawing geodesics between two arbitrary endpoints. The Geodesic Triangles visualization facilitates the application of geodesics to the Gromov conjecture [1]. In particular, the visualization draws geodesics fully connecting three vertices, then recursively draws geodesics connecting the midpoints of geodesics drawn at a previous level of recursion.

The program uses the logic from the Drawing Geodesics visualization as a subroutine, in order to draw each individual geodesic. The program lets the user step through the visualization one level at a time; at the first level, the three vertices are connected via geodesics, at the second level the midpoints of these three geodesics are connected by new geodesics, and so on. The geodesics at each level are assigned one of seven colors to make it easier for the user to differentiate which geodesics were drawn at which level of recursion.

The idea is for the mathematician using the program to see whether this technique can result in a smooth surface constructed out of non-intersecting geodesics. The user specifies the initial three vertices, and in our experience changing the values of the starting vertices can affect the behavior of the geodesics significantly.

This program has similar virtual reality controls compared to the first visualization. The user may look around using the headset and zoom in and out using the Oculus touch controllers or the arrow keys on the keyboard. To make it easier to maintain the user's sense of direction, gridlines were added to the program in Unity.

This visualization can be extended to $n > 3$ starting vertices. For instance, we extended the original program to make a "Geodesic Squares" version, i.e. a version that takes 4 vertices as input.

5 Results

The Geodesic Triangles visualization demonstrates the most formal application of our geodesic constructions, and this visualization is also built upon the logic of the first visualization. Thus, the following screenshots, published on Dr. Schikorra's webpage [2] and reproduced below, are representative of the output of the programs we developed. The screenshots are from the Geodesic Triangles program with the starting vertices of $(0, 0, 0)$, $(1, 0, 0)$, and $(0, 1, 0)$. The triangle at each of the first seven levels is shown below.

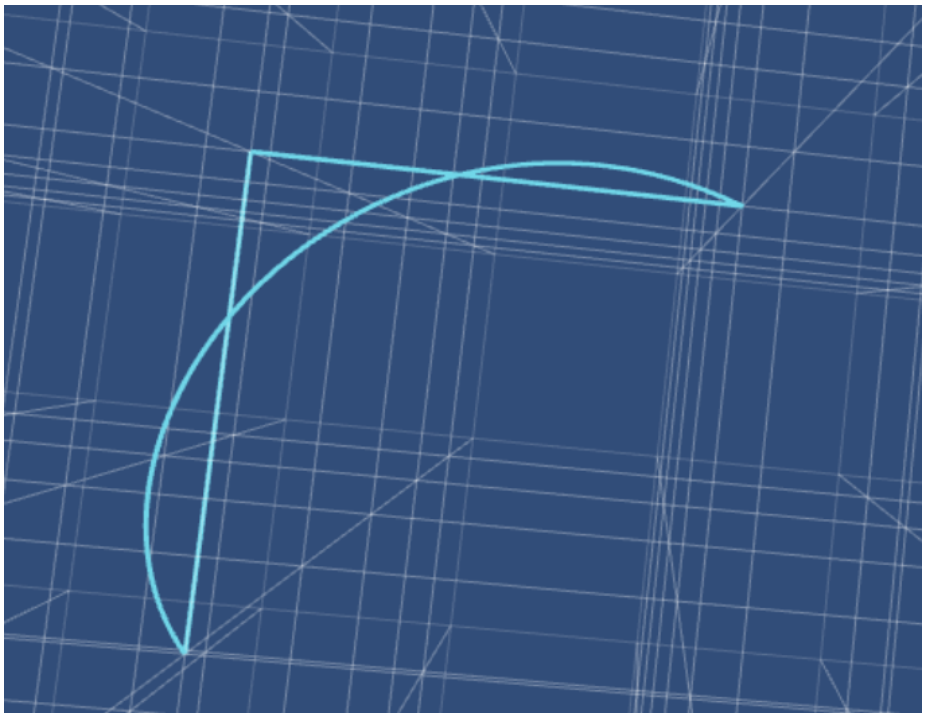

Figure 1: Geodesic Triangle at level 1. Note how the geodesics from the origin are Euclidean straight lines.

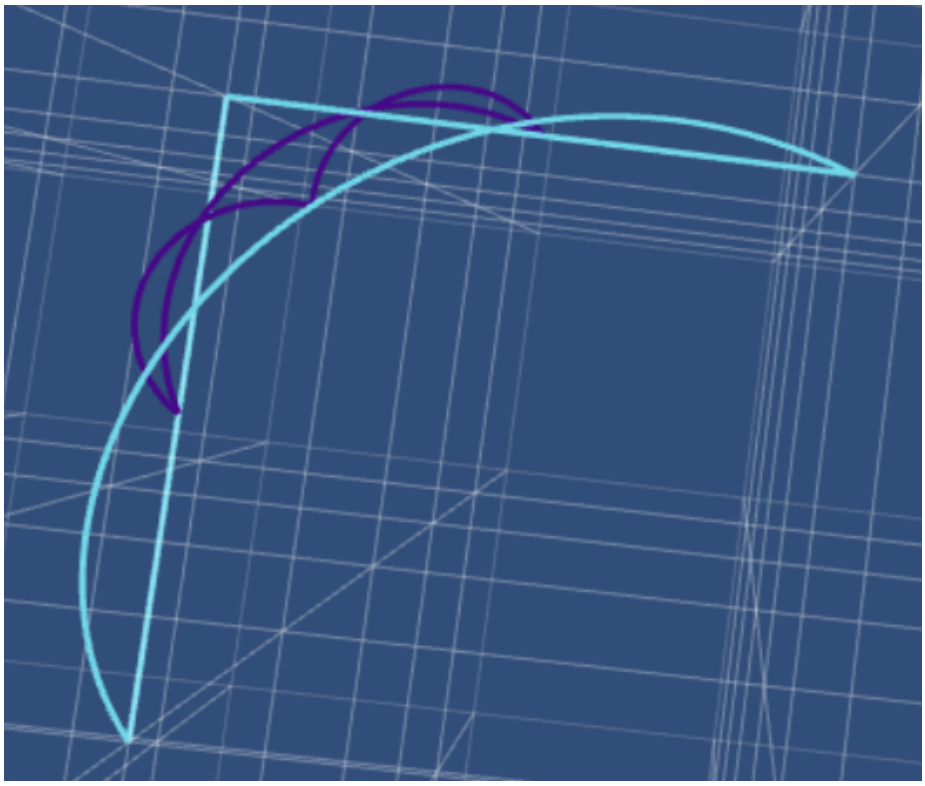

Figure 2: Geodesic Triangle at level 2. The new purple geodesics still behave regularly.

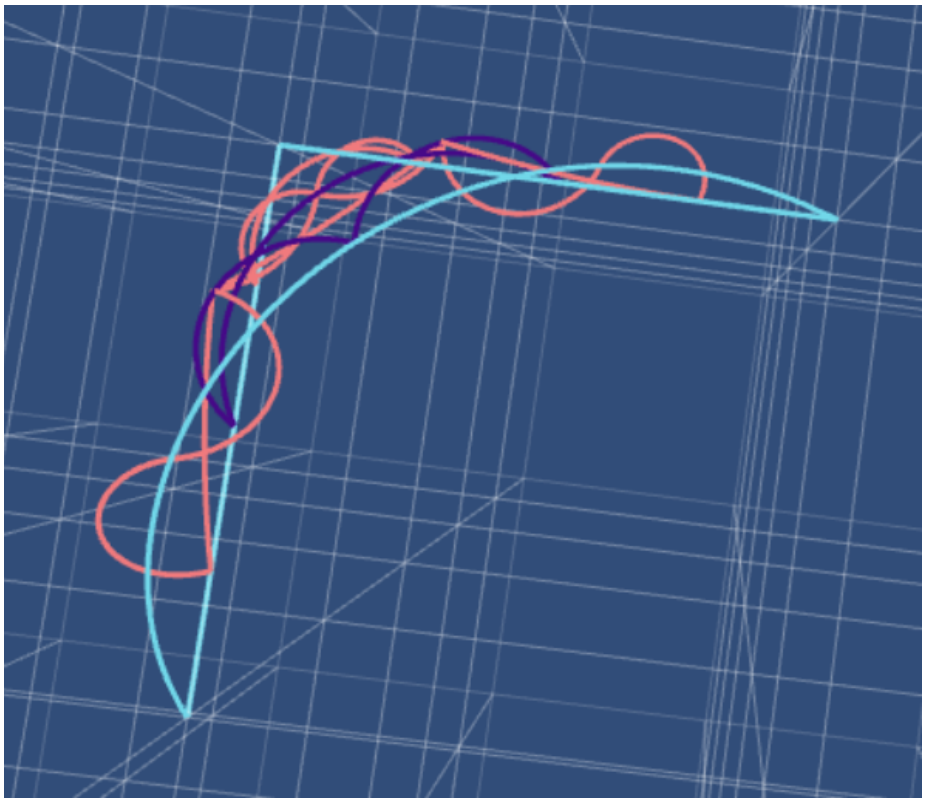

Figure 3: Geodesic Triangle at level 3. The new pink geodesics have more interesting structure than the preceding geodesics.

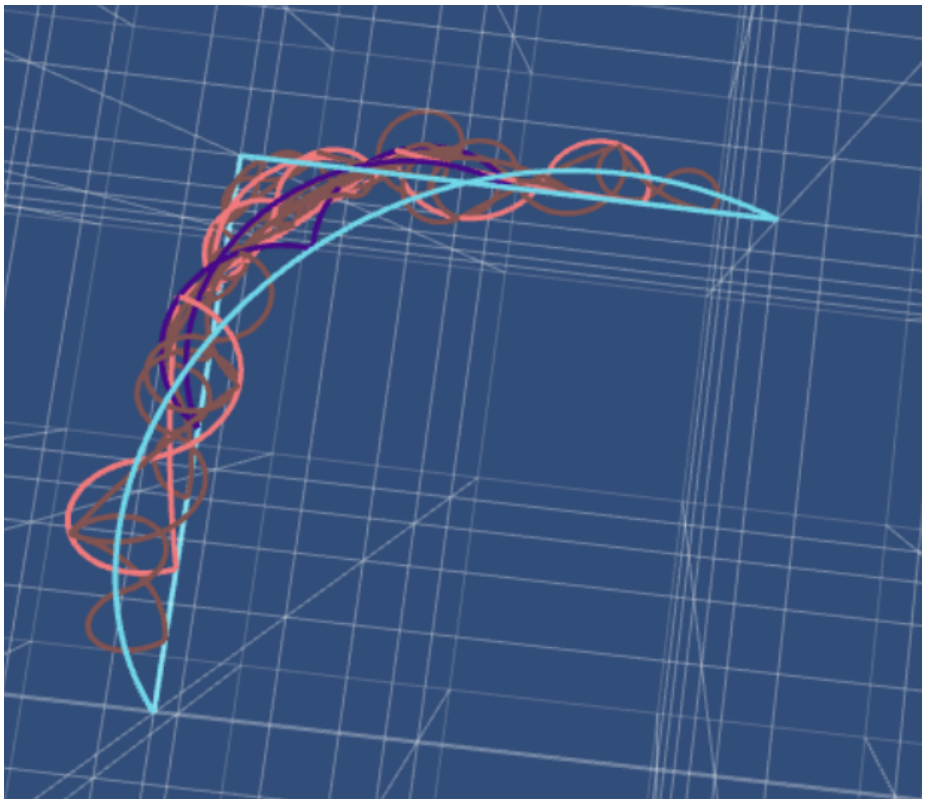

Figure 4: Geodesic Triangle at level 4. The new brown geodesics have more dramatic changes in curvature.

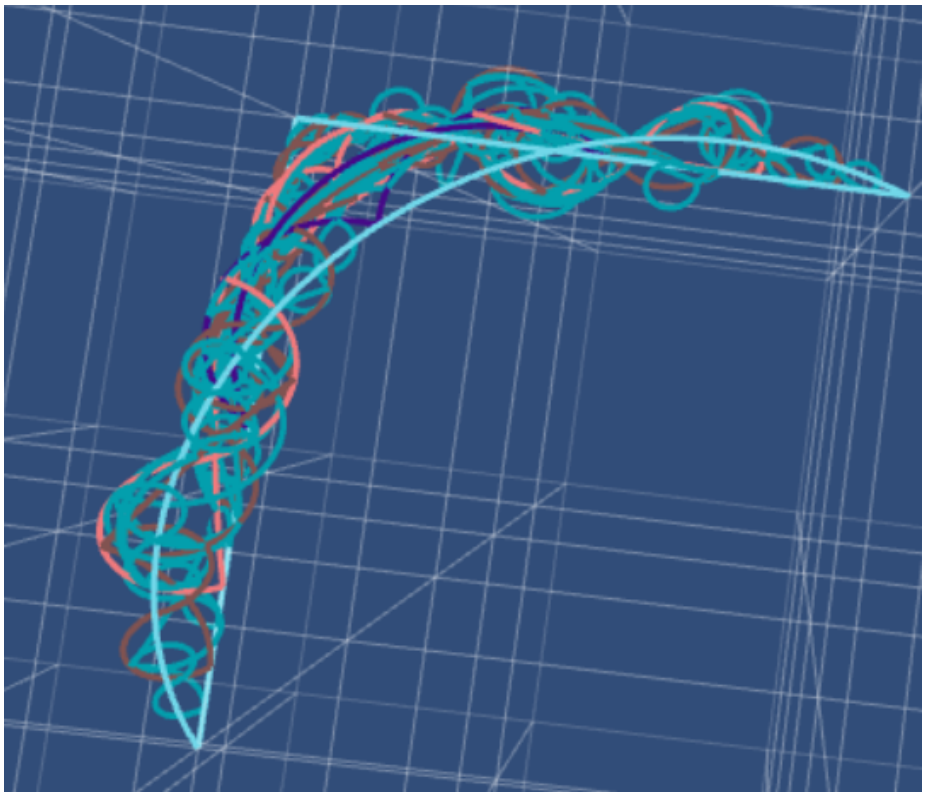

Figure 5: Geodesic Triangle at level 5. The new turquoise geodesics may possibly intersect geodesics from earlier levels, though it is unclear from this distance.

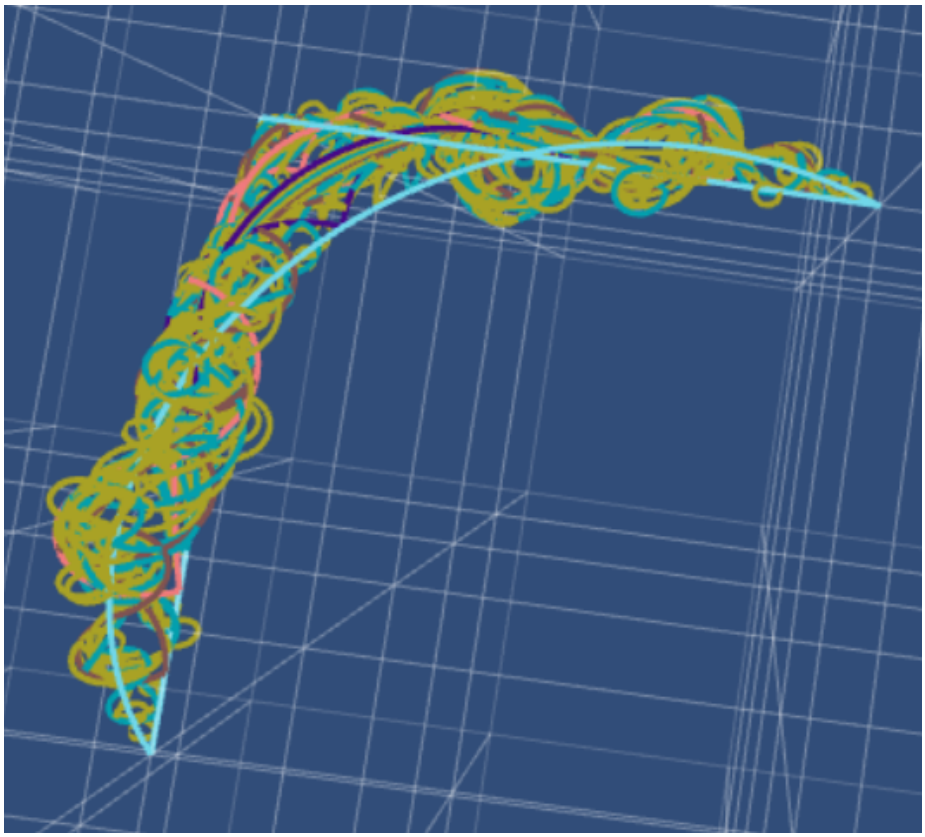

Figure 6: Geodesic Triangle at level 6. The new geodesics are colored green.

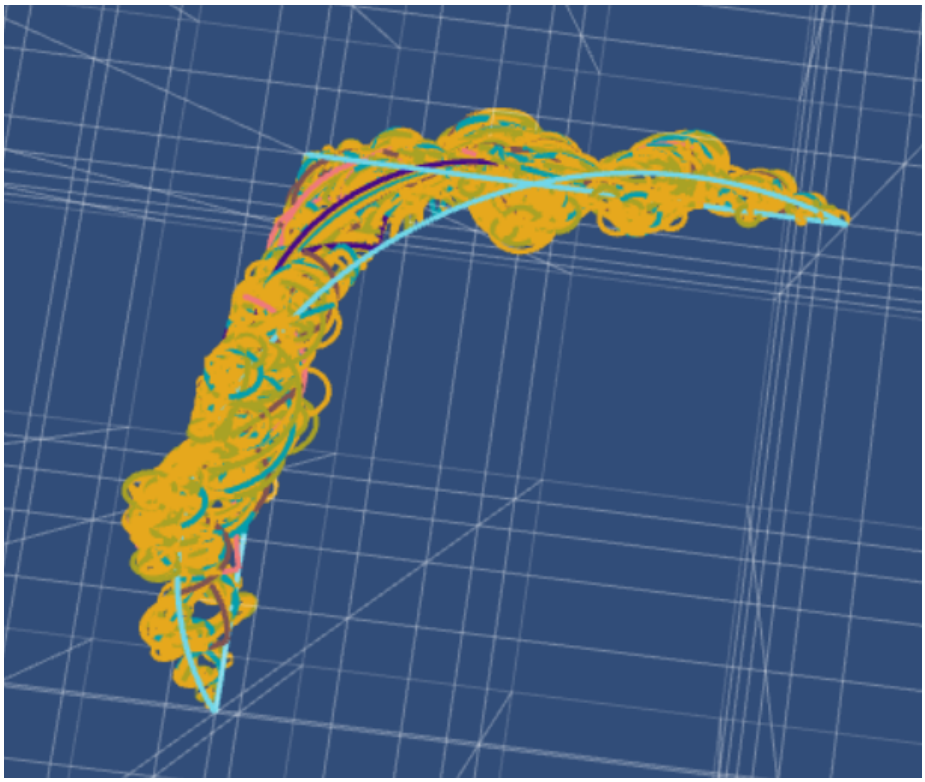

Figure 7: Geodesic Triangle at level 7. The new geodesics are colored gold. At this level and from this angle, the triangle appears quite dense though not yet a smooth surface.

6 Conclusions

Overall, we implemented three visualizations to help mathematicians develop their intuitions about the Heisenberg group and even explore how the Heisenberg group may relate to an important conjecture. We modeled 3D representations of geodesics between arbitrary endpoints in the Heisenberg group, which to the best of our knowledge is a novel accomplishment.

6.1 Future Work

Our current method for modeling geodesics likely suffers from numerical error in certain cases. For some values of T, the calculated s may have substantial error, which may lead to significant distortion of the rendered geodesic for geodesics of considerable length. One possible solution is to integrate Mathematica into our programs, especially for finding s by numerically estimating the inverse of $f(s)$ in a stable way via Mathematica. We already completed some work which

demonstrates how Mathematica functions can be called from within a Unity program, and with further development Mathematia could be called upon to improve the stability of our parameterizations of geodesics.

The use of virtual reality made interacting with these visualizations more intuitive and engaging than it would have been with a standard control scheme. With the base programs implemented, there also exists the potential for greater interaction via virtual reality. For example, many virtual reality programs allow the user to manipulate objects in 3D space by "grabbing" and pulling and pushing motions via the Oculus touch controller; our programs could be extended to allow the user to change input vertices by pulling or pushing them.

6.2 Acknowledgements

Many thanks to Dr. Schikorra for his guidance and instruction on the project. Thanks to the rest of the team who worked on this project over the past two semesters: Matthew Snyder, Isaiah Glymour, and Amy Waughen. Special thanks to Matt for his major contributions to the scripting for the Drawing Geodesics, Geodesic Triangles, and Geodesic Squares visualizations.

We are grateful for the Daimler & Benz Foundation as well as Dr. Gartside for their sponsorship of the necessary virtual reality hardware which helped make this project possible.

References

[1] Mikhael Gromov. Filling riemannian manifolds. *J. Differential Geom.*, 18(1):1–147, 1983.

[2] Armin Schikorra. Heisenberg group in virtual reality, sep 2019.